NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

MISSION EXECUTOR FOR AN AUTONOMOUS UNDERWATER VEHICLE

by

Wilfrid P. Wilkinson

September 1991

Thesis Advisor: Yuh-jeng Lee

Approved for public release; distribution is unlimited.

T259290

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Computer Science Dept. Naval Postgraduate School; 6b. OFFICE SYMBOL (if applicable): CS	7a. NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Mission Executor for an Autonomous Underwater Vehicle (U)

12. PERSONAL AUTHOR(S)
Wilkinson, Wilfrid P.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis	FROM 08/89 TO 09/91	September 1991	222

16. SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Autonomous Underwater Vehicle, AUV, CLIPS 5.0, expert system, mission execution, mission exception-handling, obstacle avoidance.

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The Naval Postgraduate School has been conducting research into the design and testing of an Autonomous Underwater Vehicle (AUV). One facet of this research is to incrementally design a software architecture and implement it in an advanced testbed, the AUV II. As part of the high level architecture, a Mission Executor is being constructed using NASA's CLIPS version 5.0. The Mission Executor is an expert system designed to oversee progress from the AUV launch point to a goal area and back to the origin. It is expected that the Executor will make informed decisions about the mission, taking into account the navigational path, the vehicle subsystems health, and the sea environment, as well as the specific mission profile which is downloaded from an offboard mission planner. Heurisitics for maneuvering, avoidance of uncharted obstacles, waypoint navigation, and reaction to emergencies(essentially the expert knowledge of a submarine captain) are required. The Mission Executor prototype, SKIPPER, attempts to do this through the use of a three-tiered reasoning system which monitors overall mission status, functional area status, and individual equipment status simultaneously.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
[X] UNCLASSIFIED/UNLIMITED [ ] SAME AS RPT. [ ] DTIC USERS	UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL: Yuh-jeng Lee	22b. TELEPHONE (Include Area Code): (408) 646-2361	22c. OFFICE SYMBOL: CS/(52)

Approved for public release; distribution is unlimited

Mission Executor for an
Autonomous Underwater Vehicle

by

Wilfrid P. Wilkinson
Lieutenant, United States Navy
B.S., United States Naval Academy, 1983

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE in COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1991

Robert B. McGhee, Chairman,
Department of Computer Science

ABSTRACT

The Naval Postgraduate School has been conducting research into the design and testing of an Autonomous Underwater Vehicle. One facet of this research is to incrementally design a software architecture and implement it in an advanced testbed, the AUV II. As part of the high level architecture, a Mission Executor is being constructed using NASA's CLIPS version 5.0. The Mission Executor is an expert system designed to oversee progress from the AUV launch point to a goal area and back to the origin. It is expected that the Executor will make informed decisions about the mission, taking into account the navigational path, the vehicle subsystems health, and the sea environment, as well as the specific mission profile which is downloaded from an offboard mission planner. Heuristics for maneuvering, avoidance of uncharted obstacles, waypoint navigation, and reaction to emergencies (essentially the expert knowledge of a submarine captain) are required. The Mission Executor prototype, SKIPPER, attempts to do this through the use of a three-tiered reasoning system which monitors overall mission status, functional area status, and vehicle equipment status simultaneously.

1 Ap513
W58833
c1

TABLE OF CONTENTS

I. INTRODUCTION 1

A. BACKGROUND 1

B. MISSION EXECUTION EXPERT SYSTEM 5

C. SCOPE OF THESIS 6

D. THESIS ORGANIZATION 7

II. CONTROL FOR AUTONOMOUS UNDERWATER VEHICLES 9

A. LAYERED CONTROL ARCHITECTURE 9

1. Massachusetts Institute of Technology Sea Grant Program 9

2. International Submarine Engineering (ISE) 10

3. The Analytic Sciences Corporation/ Naval Underwater Systems Center 12

4. Marine Systems Engineering Laboratory 15

B. HIERARCHICAL CONTROL 18

1. Intelligent Mobile Autonomous System (IMAS) 18

2. SINTEF SACOR Project 19

C. HYBRID MODELS 23

1. University of Karlsruhe Robot Project 23

2. Procedural Expert System (Esprit Project) 25

D. SUMMARY AND EVALUATION 27

III. THE C LANGUAGE INTEGRATED PRODUCTION SYSTEM 30

A. MAIN FEATURES 30

B. DEVELOPING CONTROL EXPERT SYSTEMS IN CLIPS 35

C. COMPARISONS AND BENCHMARK 37

D. PORTABILITY 39

IV. ONBOARD INFORMATION PROCESSING 41

A. DOWNLOADING POSTURES AND COMMANDS FROM THE MISSION PLANNER 41

B. UPDATING FROM THE OBSTACLE AVOIDANCE DECISION MAKER 43

C. UPDATING FROM THE SONAR MODULE 44

D. INTERFACE WITH THE REPLANNER 45

E. UPDATING FROM THE VEHICLE CONDITION MONITOR 46

F. INTERFACE WITH THE GUIDANCE SUBSYSTEM 47

V. DESIGN OF THE PROTOTYPE EXPERT SYSTEM 49

A. PHILOSOPHY OF DESIGN: REASONING ABOUT SEVERAL WORLDS 49

B. SEQUENCE OF CONTROL 56

C. TRUTH MAINTENANCE AND THE ROLE OF UNCERTAINTY . 64

1. Maintaining a Consistent Knowledge Base 64

2. Uncertainty 67

D. MISSION DOCUMENTATION AND OBJECT PERSISTENCE . . . 68

1. The Need for High Level Mission Documentation. 68

2. Object and Fact Persistence in the Executor. 69

VI. PROTOTYPE IMPLEMENTATION AND SIMULATION 70

A. CONTROL CONSTRUCTS AND OBJECT IMPLEMENTATIONS . 70

B. LAYERING OF RULES 77

C. USE OF FUZZY LOGIC AND TRUTH MAINTENANCE 79

D. RESULTS 82

E. EVALUATION 88

VII. CONCLUSION AND RECOMMENDATIONS 90

A. SUMMARY OF RESULTS AND CONTRIBUTIONS. 90

1. A Prototype Expert System for Mission Execution 90

2. Software Architecture for Mission Execution 90

3. Determination of Guidance Interrupt Commands 91

4. Identification of New Data Flow in the Baseline System 91

B. FUTURE WORK 91

1. Mission Executor Portability 91

2. Interfacing the Executor to Dependent Modules 92

3. Porting the Executor to the AUV II Graphical Simulator 92

4. Incorporation of Specialized Mission Rules 93

LIST OF REFERENCES 94

APPENDIX A. MISSION EXECUTOR SOURCE CODE 99

APPENDIX B. TESTING SCENARIOS 144

INITIAL DISTRIBUTION LIST 211

LIST OF TABLES

Table 5-1. AUV Obstacle Avoidance Maneuvers 60

Table 5-2. Executor Commands to Guidance 66

Table 6-1 Scenario Results. 85

LIST OF FIGURES

Figure 1-1. NPS Baseline AUV System 4

Figure 2-1. MIT'S Layered Control Architecture 11

Figure 2-2. TASC/NUSC AUV Software Architecture 14

Figure 2-3. MSEL EAV III Software Architecture 17

Figure 2-4. IMAS Software Hierarchy 20

Figure 2-5. Module Interfaces in SACOR 22

Figure 2-6. Software Control in the Karlsruhe Robot 26

Figure 3-1. A Sample CLIPS Rule 32

Figure 3-2. Use of the CLIPS Truth Maintenance Construct *logical* 33

Figure 4-1. Module Interfaces in NPS AUV II Software System 42

Figure 4-2. System Monitor Objects 48

Figure 5-1. Naval Underwater Systems Center Matrix 52

Figure 5-2. Event and Demand Driven Data 53

Figure 5-3. Overall Mission Executor Schema 55

Figure 5-4. Functional Area Hierarchy 57

Figure 5-5. Situational Awareness Through Salience 65

Figure 6-1. Overall Mission Assessment Rule 80

Figure 6-2. NPS Pool Mission Schematic 84

ACKNOWLEDGEMENTS

I thank Dr. Yuh-jeng Lee for his patience, encouragement, and support during the development of this thesis. He provided liberal guidance while allowing me to independently explore several areas related to expert systems and robotics. He has been a true mentor.

I wish also to thank Commander Gary Hughes, USN, for his help in correctly drafting my research.

Most importantly, I thank my lovely wife, Mary, for all of her support and patience during the 24 months at the Naval Postgraduate School. Without her support, this thesis would not have been possible.

I. INTRODUCTION

A. BACKGROUND

The development of autonomous underwater vehicles has been an ambition for decades. Only recently, however, have practical autonomous underwater vehicles appeared to be reality. Since the development of SPURV I (one of the first autonomous underwater vehicles in the United States) at the University of Washington's Applied Physics Laboratory in 1963, government and civil interest has been fueled by the potential for many applications (Busby 90, p. 65). The hope is that the control system for the vehicle will adequately perform the man-machine interaction that regularly takes place on manned submersibles. Military interest over the last decade has increased, particularly with the advent of tactical automated weapons and air reconnaissance vehicles. Recent events in the Gulf War have validated the advances in automated weapons during the 1980's. As Vice Admiral Stanley Arthur, Commander U. S. Naval Forces Central Command during Operation Desert Storm, remarked (on Tomahawk cruise missile system effectiveness):

> ... target-arrival percentages look good. When dealing with a system such as Tomahawk, all the details can be planned carefully. Then when the missile is fired, the electronic gizmos take over. These integrated circuits do not get scared; they do not forget; they follow orders well. The critics--who said Tomahawk would work only on a single test range and that it would get lost in the desert--were wrong. News reports seem to support the idea that attacks by robots have a unique psychological effect on people. (Arthur, 1991, pp. 85-86)

On the effectiveness of the remotely piloted vehicle, Pioneer I, Vice Admiral Arthur also observed:

> Remotely piloted vehicles proved to be marvelous, versatile devices. They allowed the battleships to attack the enemy on their own, without the need for outside assistance in spotting. Spotting by the RPV's not only allowed for accurate naval gunfire support, but also provided instant battle damage assessment. The RPV offers quick response and flexibility, because it is under positive tactical control and has the ability to get below a low ceiling. Of course, the highlight of the war for the RPV has to be the incident in which a remotely piloted vehicle flew over Iraqi troops. By that time, the Iraqis knew what would be coming next, so they surrendered to the RPV--presumably the first occasion in the history of warfare for human beings to capitulate to a robot. (Arthur 1991, p. 86)

Several marine autonomous and remotely-piloted vehicles are already in use for such diverse functions as underwater cable inspection, hydrography, and mine-hunting. The practical advantage of low-risk to human operators coupled with the potential ability to operate at over-the-horizon distances make the autonomous underwater vehicle a highly desirable project. Although development of autonomous underwater vehicles has progressed more slowly than the well-publicized air and land vehicles, advances during the 1980's in artificial intelligence and robotics have proven to be monumental. As a consequence, the Defense Advanced Research Projects Agency (DARPA) has been the primary major funding source for the evolutionary advances made during the last decade. (Polmar, 1991, pp. 122-123). Early research in autonomous underwater vehicles at the Naval Postgraduate School centered around computer and control surface interfaces tested in the first testbed, Autonomous Underwater Vehicle I (AUV I), a tele-operated underwater robot. Efforts since that testing ended have focussed on an entirely autonomous vehicle, Autonomous Underwater Vehicle II (AUV II).

Previous student theses at the Naval Postgraduate School have primarily concentrated on the use of artificial intelligence in mission-planning and guidance control of the vehicle. Cloutier investigated and developed a vehicle Guidance subsystem. His subsystem provides fora proper vehicle configuration for path following from waypoint to waypoint (Cloutier 1990). Ong researched and developed an extensive offboard Mission Planning expert system. This was incorporated into an advanced simulator developed previously (Ong 1990). MacPherson studied rule-based control of an AUV. He implemented this control system in a simulator written in LISP under the Knowledge Engineering Environment (KEE). Generic mission templates were developed for various specialized mission profiles (MacPherson 1988).

The current generation of student theses attempts to take the development of an intelligent control system for the AUV into the next increment of evolution. The baseline diagram of the projected software system is depicted in **Figure 1-1**. Both intermediate level modules (such as the pattern recognition and navigation software) and high-level modules such as the mission planner/replanner and mission executor are now in development. Central to the high-level control is the Mission Executor described in the next section. An advanced decisionmaking capability is needed to make an autonomous underwater vehicle (AUV) truly adaptive and survivable. The noted naval analyst Norman Polmar recently surveyed the current advances and underscored the demand for intelligent capability in vehicle technologies :

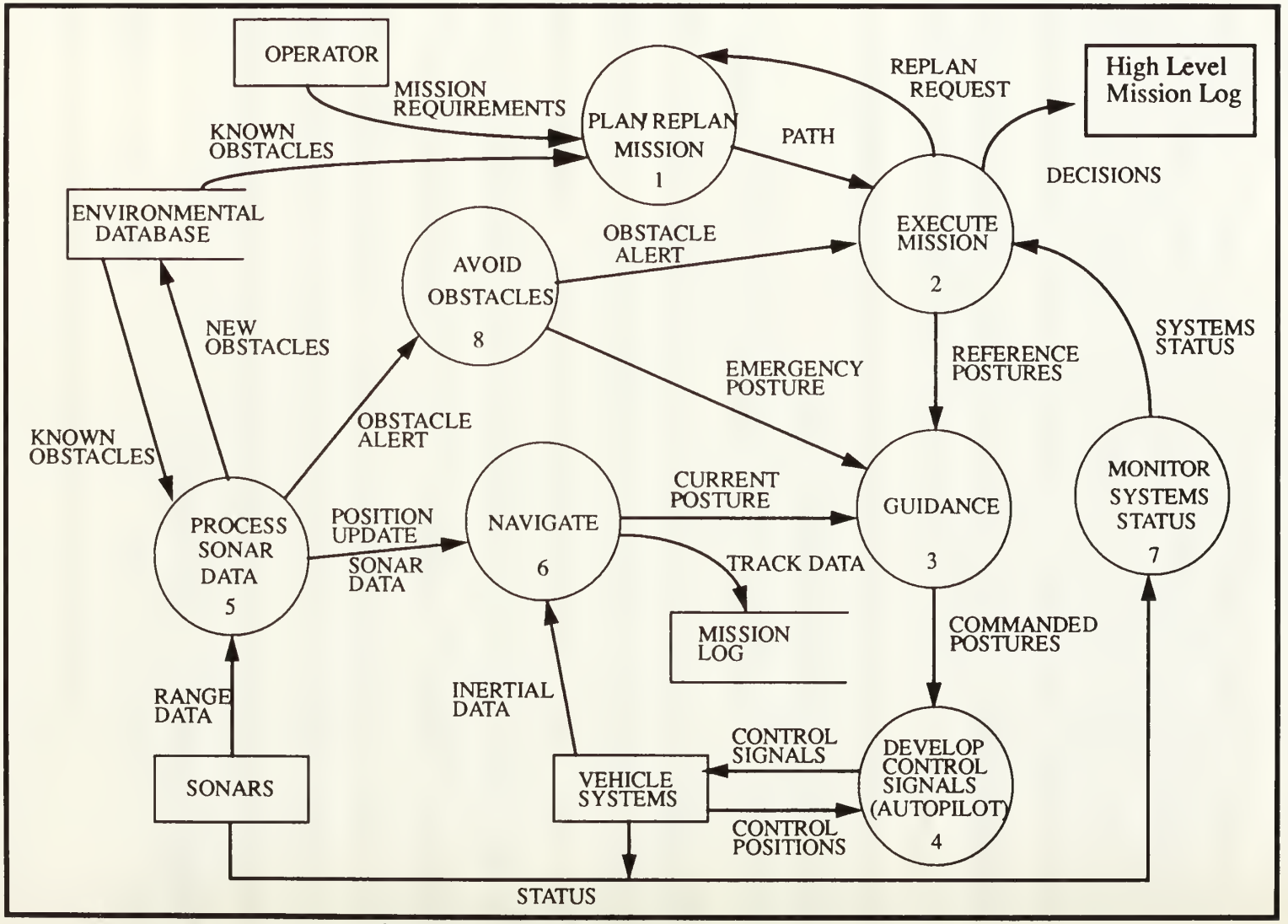


Figure 1-1 . NPS AUV II Baseline Data Flow Diagram

> The key to success in all the testing will be component or "enabling" technologies: navigation, composite hull materials, guidance, energy source, propulsion, communication links, and signal processing as well as the specific mission packages. Advanced autonomous underwater vehicles will require enhanced sensor and decision-making capability. (Polmar 1991, p. 123)

B. MISSION EXECUTION EXPERT SYSTEM

The control architecture of AUV II has undergone several phases of development. Many methods of autonomous control are being used in vehicle testbeds around the country. Some allow for layering of control in the vehicle while others maintain a more traditional horizontal model of planning, execution and analysis. One general architecture that has recently come of age is top-down flow of control ranging from Strategic level control through tactical level to the low-level monitoring level (the level at which vehicle software and hardware actually interface). Higher levels of abstraction perform some of the activities (some time-sensitive) which require measured decision-making.

The Naval Postgraduate School's AUV control structure has undergone an evolutionary development. Control in AUV I and early control structures in AUV II was essentially low level closed-loop. Incremental changes to the software design in 1990 necessitated the integration of a Mission Executor to integrate and coordinate intelligent waypoint following and obstacle-avoidance. The Mission Executor functions involve continuous real-time analysis and high-level supervision of vehicle systems throughout the life of a mission. Thus the Executor must make real-time decisions, often in an environment of uncertainty or incomplete knowledge (Healy 1990a, pp. 177-183). While not all situations can be completely provided for in the system, the ambition is to design

heuristics which make it possible for the Executor to deal with extensions of well-known problems.

C. SCOPE OF THESIS

The Mission Executor, in the broadest sense, must be able to safely control movement between a mission starting point and a mission goal. In doing so, it must operate between three models: that of the mission world, the vehicle world and the environmental world. To supervise the vehicle world suggests that the Executor must monitor and control vehicle "health" such as battery state, internal system pressure, and temperature. It must also provide for response to deteriorating condition of the vehicle sonar, navigation system, or guidance systems. The loss of a major onboard equipment such as the sonar or navigation systems would probably be catastrophic and would at least result in a mission degradation. The Executor must supervise the subsystem recovery procedure or make decisions that can circumvent the problem. Failing that it must make a strategic-level decision to abort the mission.

Control of the vehicle in the context of the environmental world means reaction to topological features such as undersea terrain and obstacles (both moving and non-moving), a significant change in atmospheric conditions, or any external threat which would physically hinder the vehicle from making the transit to the goal point.

Monitoring of the mission world entails awareness of transition points between normal transit and beginning a special mission profile. Possible speed/depth changes, special requirements for inshore navigation, and deployment of any equipment must be

considered. Most importantly, the mission priority must be balanced against vehicle survival and reusability. Heuristics for this must be incorporated in the software.

D. THESIS ORGANIZATION

Chapter II is a survey of previous work on AUV control systems and related technology. Current AUV software control systems at many different research facilities are discussed. AUV research is classified by the types of software architecture.

Chapter III is a feasibility study of using the C Language Integrated Production System (CLIPS) version 5.0 expert system tool as the development environment for the Mission Executor. This chapter also includes analysis of the portability of CLIPS to GESPAC, the AUV II's onboard computer.

Chapter IV is a description of onboard information processing. It details the interactions between various modules of the software architecture outlined in the baseline diagram, Figure 1-1.

Chapter V is a description of the prototype expert system architecture from a theoretical stance. The development of layers of reasoning in software is highlighted. Issues such as the proper combination of rules and objects, the role of uncertainty and truth maintenance, and knowledge-database object persistence are discussed in the context of the Autonomous Underwater Vehicle. Specific software constructs are left to Chapter VI.

Chapter VI is a description of both the Mission Executor constructs and the Executor simulation. Rules which incorporate some special complexity or feature are described in detail.

Chapter VII outlines contributions, conclusions and extensions for further work.

II. CONTROL FOR AUTONOMOUS UNDERWATER VEHICLES

This chapter is an overview of Autonomous Vehicle high-level control development at other institutions and commercial organizations. The various autonomous vehicle programs are classified by software architecture. Differences and similarities to the Naval Postgraduate School's testbed AUV II are discussed in the concluding summary.

A. LAYERED CONTROL ARCHITECTURE

1. Massachusetts Institute of Technology Sea Grant Program

Bellingham and Consi of the Massachusetts Institute of Technology have been at the forefront of AUV research for the last several years. The MIT program has worked with Charles Stark Draper Laboratories and International Submarine Engineering on the development of Sea Squirt I (Bellingham 1990, p. 23). This platform uses a Motorola 68020 processor. MIT Sea Grant is implementing a software architecture based on Brook's layered control architecture (Brooks 1986, pp. 365-372). This architecture is behavior-oriented, using the subsumption approach. The objective is to move away from the traditional robot architectures which require a world model be incorporated. This is due to the AUV compaction problem: a small submersible cannot support high-resolution sonar or an extensive, intelligent vision system. Consi and Bellingham argue that the world model is then severely flawed, which may lead to incorrect or conflicting behaviors (Bellingham 1990, pp. 23-24). In the subsumption model, high-level behaviors include

planning and monitoring while lower level behaviors are oriented toward the reflexive states. The software development itself is intriguing. Low-order behaviors are first installed and verified in the testbed. When satisfactory performance is achieved, the next level of complex behaviors is then added. Abstractly, the lower level is subsumed by the higher level, but nonetheless carries out its behaviors in real-time. The architecture is designed to be reconfigurable for different missions. (Bellingham 1990, pp. 24,27)

Despite an initial retreat from the world model paradigm, the MIT group believes it might be useful in complex missions to incorporate world modeling into high layers. This would free lower levels to continue to operate in real-time as they must. The overall architecture will become distributed for sensor processing. (Bellingham 1990a, pp. 75-78) A diagram of the basic behavior layering is shown in **Figure 2-1.**

2. International Submarine Engineering (ISE)

International Submarine Engineering (ISE) is currently cooperating with MIT on the Sea Squirt research. International Submarine has previously developed several software architectures for its series of ARCS underwater vehicles (Zheng 1990, p. 71). Original work focussed on a software architecture based on the Navy watch team concept of functionality. Control was based on the Cooperating Experts Paradigm, in which separate modules for piloting, independent transit and collision avoidance all worked to form a fused plan. After much experimentation, this was discarded as infeasible because module functionality did not always correspond well to the many tasks that even one human carried out. Further decomposition was necessary.

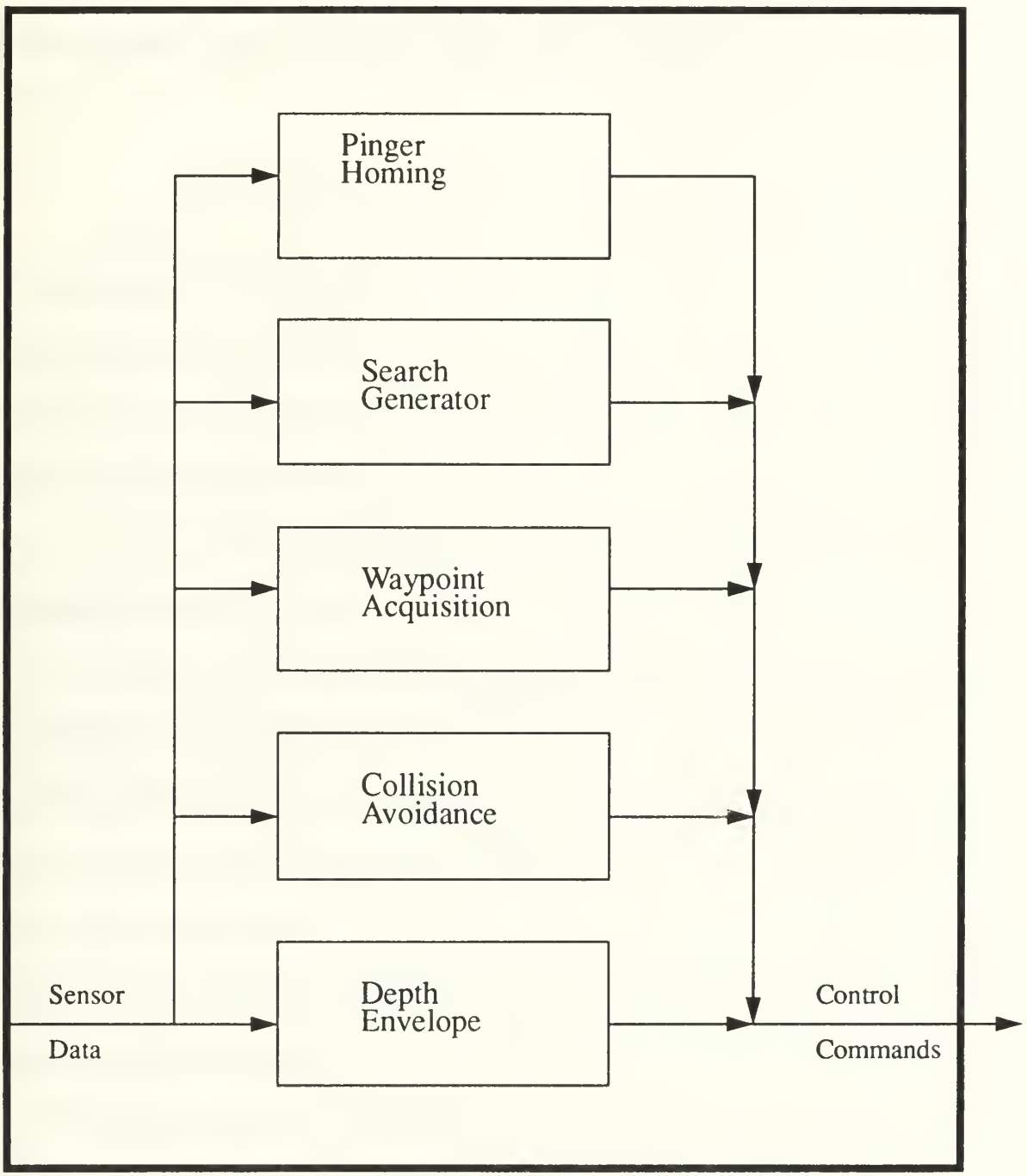


Figure 2-1. MIT's Layered Control Architecture

ISE's new architecture is object-oriented and behavior-oriented, based upon Brook's seminal layered control architecture of the mid-1980's. ISE incorporates rule-based heuristics and learning through reflexive behaviors, logical behaviors and learned behaviors.

3. The Analytic Sciences Corporation/ Naval Underwater Systems Center

The Analytic Sciences Corporation (TASC) and the Naval Underwater Systems Center (NUSC) have developed a novel software architecture which combines aspects of real-time layering, functional decomposition and subsumption. It is a new structuring of the traditional perception, analysis and action paradigm of robotic software. The software architecture is being implemented in C++. (Schudy 1990, p. 9)

Unlike the division of functions in the Intelligent Mobile Autonomous System (IMAS) in which each level carries a similar structure for conflict-resolver, world model and level-specific function, the division of tasks in the NUSC/TASC architecture is non-homogeneous both horizontally and vertically. It is divided horizontally into an analysis hierarchy which is composed vertically of increasingly competent levels of assessment. The bottom level is real-time while the event assessment at the highest level is decidedly non-real-time. This hierarchy at each level functions as effectors for the tightly-coupled planning and supervision sections of the Task Decomposition hierarchy. The planning section consists in the levels of mission planning (highest), phase planning, task planning, and action planning (lowest level). The supervision functional section is divided into mission level plan execution (highest), phase level plan execution, task execution, and

subsystem supervision (lowest). Positioned between the analysis and planning areas is a response system in which responses are merged and subsumption of behaviors occurs. These hierarchies are separated from the low-level functions of sensory data, internal monitoring and guidance control. (Schudy 1990, pp. 10-14) This is depicted in **Figure 2-2** .

Rather than just consider the division of function by functional level, there is also decomposition by time. Real-time control only encompasses the lower levels, monitoring and control in the most atomic sense and the planning/assessment that is one level above that. The actual flow of control is very evident. The advantages of such a system are that mission execution can be monitored at a high rate for low level behaviors while, as in layered control, the high level behaviors such as planning and global assessment are done at a less time-constrained rate. (Schudy 1990, pp. 13-14)

Unlike the strict layered control hierarchy, this system maintains a detailed world model which consists of a vehicle internal model, an environmental model, and a event assessment model. Like the layered control hierarchy, there is subsumption. Rather than describing it in terms of competent behaviors, it is described in terms of assessment and response. Assessment modules describe behavior in mathematical models (Schudy 1990, pp. 14-16). Response modules are intermediate to the planning modules and incorporate behavioral alternatives.

Mission execution is carefully supervised by an overall mission execution manager. In one sense, the overall mission execution manager is nothing more than a high-level sequencer. The mission execution manager in turn supervises phase execution

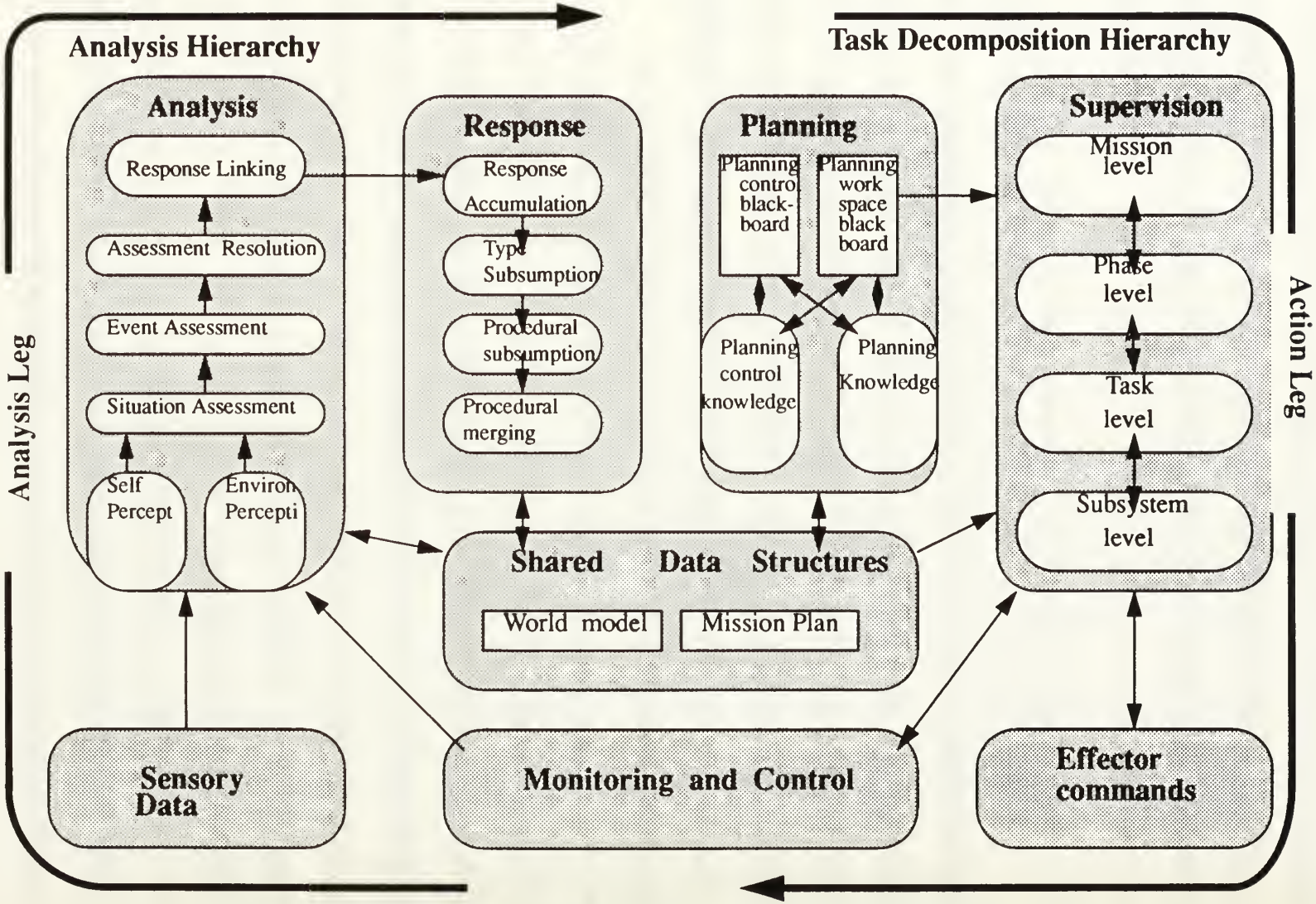


Figure 2-2 . TASC/NUSC AUV Software Architecture

managers. Phase execution managers have responsibility for monitoring an entire phase of the mission. These are intermediate mission executors which oversee the task execution managers. In naval terms, the task execution manager can be described as a special detail. It is the task execution manager's responsibility to ensure that a special evolution such as turning at a waypoint is carried out. Further, the task execution manager monitors a subsystem manager for each of the following environmental sensing, navigation, guidance, communication and energy. (Schudy 1990, pp. 18-20) This software architecture is one of the few to specifically mention mission execution as a high-level control activity.

4. Marine Systems Engineering Laboratory

Marine Systems Engineering Laboratory at the University of New Hampshire has been involved in AUV research for over fourteen years. The first underwater autonomous vehicle developed was EAVE I (*E*xperimental *A*utonomous *VE*hicle I) which was completed in 1978. It was designed for cleaning underwater pipelines. In 1986, MSEL was given a charter to develop knowledge-based AUV's which could render complex decisions and operate independently. (Thus, the acronym for EAVE became KB/EAVE.) EAVE is a larger class of AUV than the NPS AUV II (and similar small AUV's) which is hardware-intensive: resident onboard are three Motorola 68000 processors for the lower level and VME 68020's for the higher level decision making. Lowest level control, guidance and monitoring functions are carried out in the lower level 68000 processors. (Blidberg 1990, pp. 33-34)

Although the upper and lower levels of decision-making are coupled, MSEL designed the lower level to be stand-alone in the event that strata independence was necessary. The design of the KB/EAVE software system for the EAVE III generation of vehicles is structured around data that is transformed from raw sensory output eventually to knowledge for complex decision-making. This is achieved through functional layering. Mission functions reside at the highest level while control functions are at the lowest level. This design is not wholly hierarchical in the sense that each level is divided horizontally into data manipulation on one side and control on the other. This design is depicted in **Figure 2-3.** The hierarchical division is based on time constraints. As in many of the control architectures, the notion is to give the planning and assessment functions more time while requiring guidance and direction motion control to operate quickly. (Blidberg 1990, pp. 35-36)

The lowest level reads and controls sensors and activates control surfaces. In the next higher level, the system level receives packaged data from the lowest level and generates intermediate level commands. The environment level (just above) performs navigation functions and planning based on goals received from the mission (highest) level. This level performs the tasking and uses the state of the vehicle at the environment level to generate high-level plans. A philosophy that the system can artificially evolve has prompted MSEL to attempt to build and test the lower level before it proceeds to the next highest level (Blidberg 1990, pp. 36-37). This is similar in concept to construction in Brooks's layered control architecture.

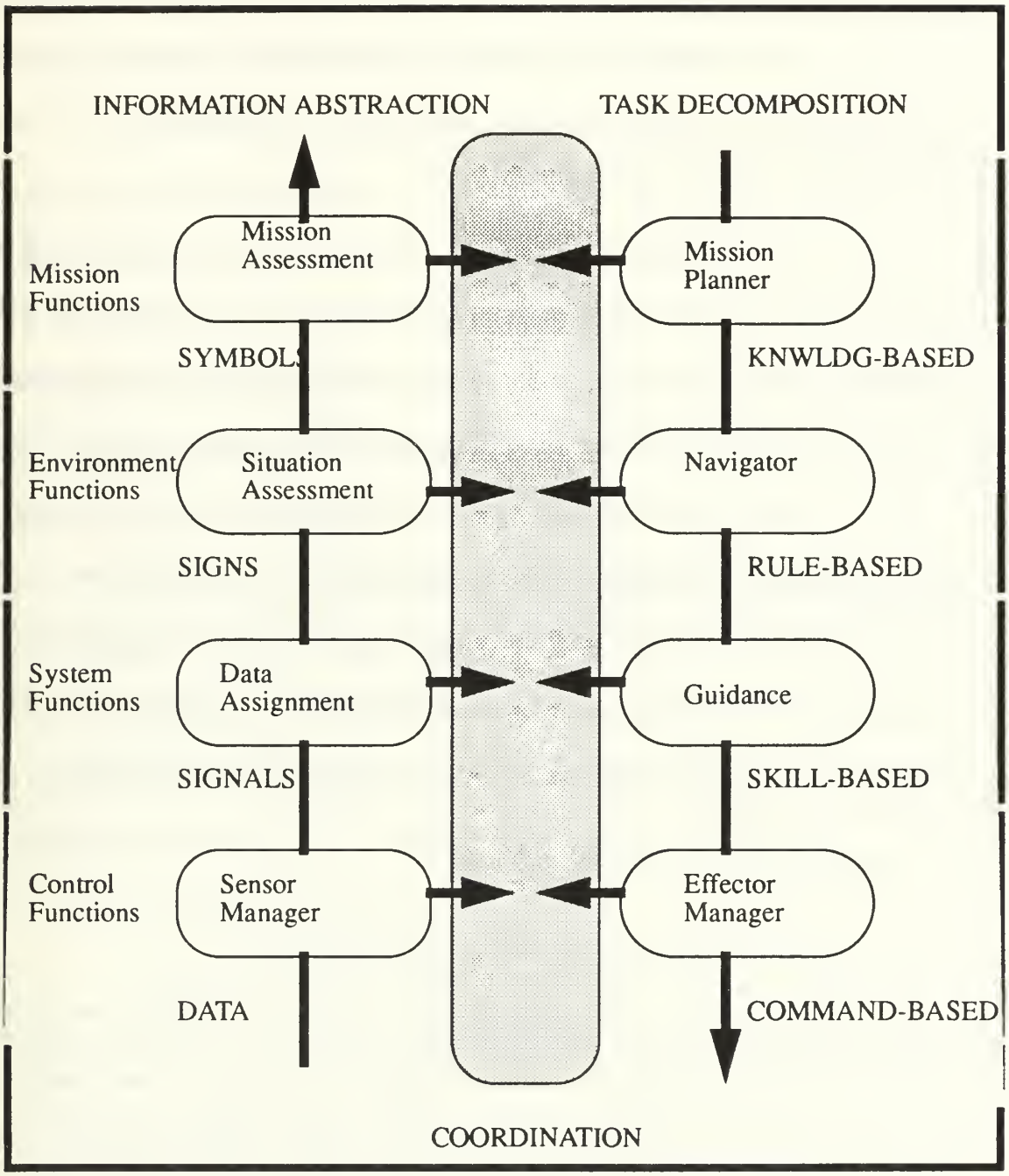


Figure 2-3. EAV III Software Architecture

The decision-making software in the higher levels uses what is known as schema-based reasoning developed by Turner of the University of New Hampshire MSEL group. These schema are essentially templates of reasoning and behavior which cover such areas as reaction to critical situations, development and consideration of plans and focus of attention (Blidberg 1990, pp. 39-41).

The MSEL KB/EAVE software development also involves using the Portable Common LISP Subset or PCLS. The effort to find a portable object-oriented LISP subset was based on a need to find a programming environment that was independent of hardware layout. While the C language is being used for numerically-intensive tasks such as sensor data processing and guidance tasks in the two lower levels, intermediate and high-level functions are targeted for development in PCLS. Part of the world model (navigation/situation assessment level) is already functioning in the testbed. PCLS works well because it does not have the temporal overhead usually associated with LISP. MSEL describes it as "garbage collection compaction. " (Bowen 1990, pp. 221-226)

B. HIERARCHICAL CONTROL

1. Intelligent Mobile Autonomous System (IMAS)

Meystel of the Laboratory of Applied Machine Intelligence and Robotics (LAMIR) of Drexel University and Isik of Syracuse University of have developed a hierarchical model of control for a terrestrial robot vehicle under development for the Belvoir Army Research and Development Engineering Center (Isik 1990, pp. 241-242). Although this involves a wheeled surface vehicle with a vision sensor system, the Nest

Hierarchical Control paradigm is applicable for subsea autonomous robots with sound ranging sensors. It provides both sufficient redundancy and layering of automated reasoning, as the following description and **Figure 2-4** suggest.

The software is divided hierarchically into the planner, navigator, pilot, and actuator/controller levels. Each level has its own separate sensor bank for perception, a map for world model reasoning, and a reporter for intelligent control. The functional unit itself (planner, navigator, pilot and actuator/controller) has its own database, rulebase and evaluator. Each stratum has a different level of resolution for its sensors. Data conflicts are resolved via what is known as resolution relevance. The Reporter module in each stratum performs the conflict resolution. (Isik 1990, p. 242)

The rule base is modeled as a production system. Fuzzy set theory is used in the controllers to describe relationships and control actions within and outside of the vehicle. The global view of the environment via the vision system is used in the top two layers while the Pilot level uses a local or "windshield" view to guide the vehicle along the planned path (Isik 1990, p. 242-243).

2. SINTEF SACOR Project

SINTEF Automatic Control of Trondheim, Norway has developed several robotic vehicles over the past several years. The current vehicle being used is the SPRINT 101, a tethered vehicle. This is a data-autonomous vehicle with six sonars which receives power via an umbilical cable. The software resides on a 68020 processor.

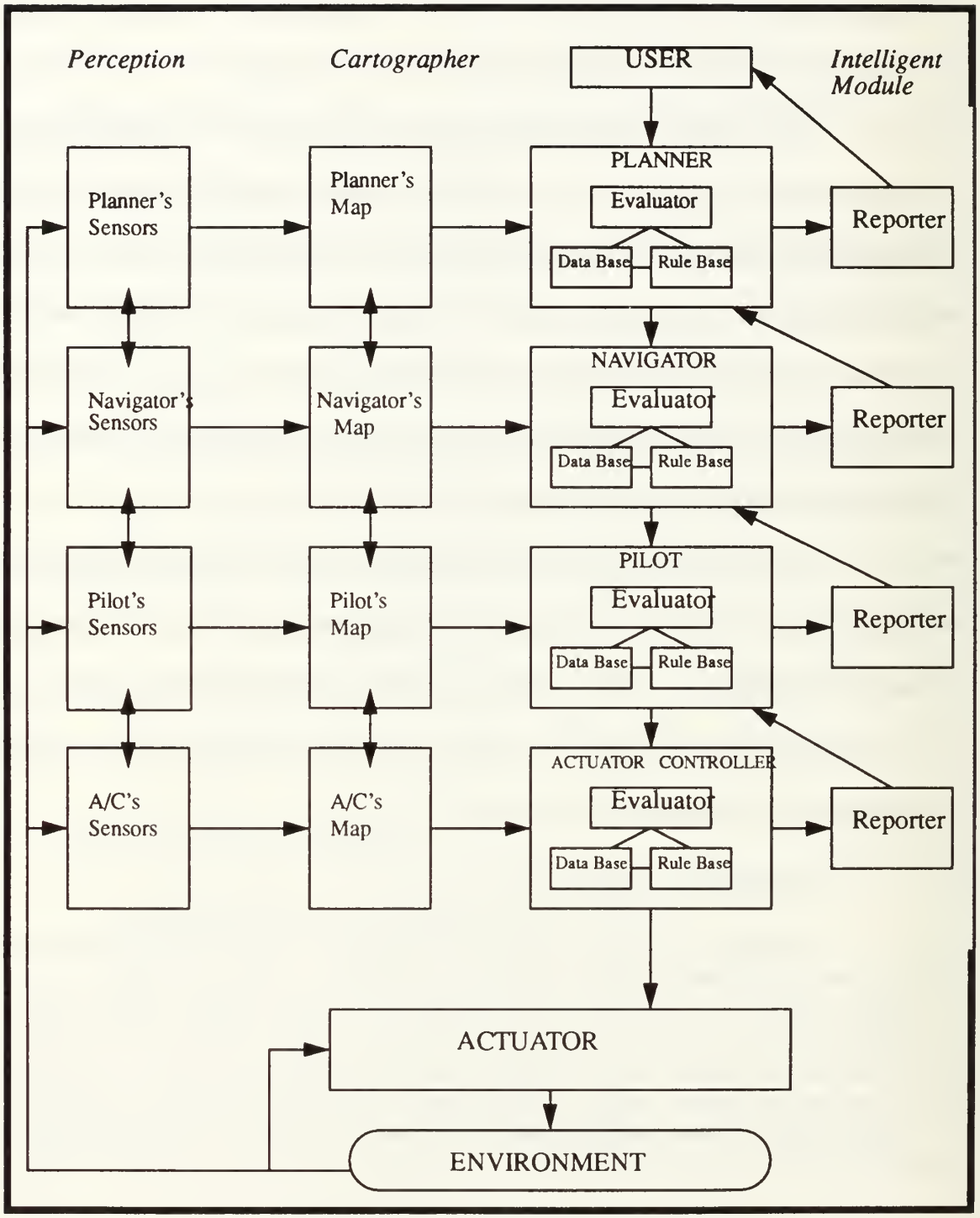


Figure 2-4. IMAS Software Hierarchy

SACOR is an acronym for Software for Autonomous Control of ROV90. Software is being developed in C++ and currently resides on SUN workstations (Rodseth 1990, pp. 15-18).

SACOR is actually a software design in two parts. The administrative section, known as ASACS (Administrative System for AUV Control Software), sequences and controls the flow of data in the system. The software is object-oriented. Modules, which are abstracted behind data structures, cannot communicate directly. They must pass data through strict interfaces. This is principally the object-oriented paradigm. ASACS is essentially a hierarchical system. The database controller interfaces modules to state variables. Progress in status is compared to desired goals. An event handler generates an object for each event and schedules it for transport to the correct module. A monitoring unit known as the Watchdog conducts error checking of vehicle internals and navigational progress. The Captain module is a simple sequencer for the mission plan. The plan itself is a hierarchical structure of state variables and conditions under which they are activated (Rodseth 1990, pp. 15-17). The dataflow and control is diagrammed in **Figure 2-5**.

Modules are either update or action modules. Action modules channel commands from the highest levels down. Modules on lower levels outweigh those in higher levels. (Presumably this is because lower level modules are real-time directors of action.) New goals are developed through plan conflict resolution. Update modules provide information from sensors attached to actuators and may direct action across a range of state variables (Rodseth 1990, pp. 18-20). Rodseth's description of the current

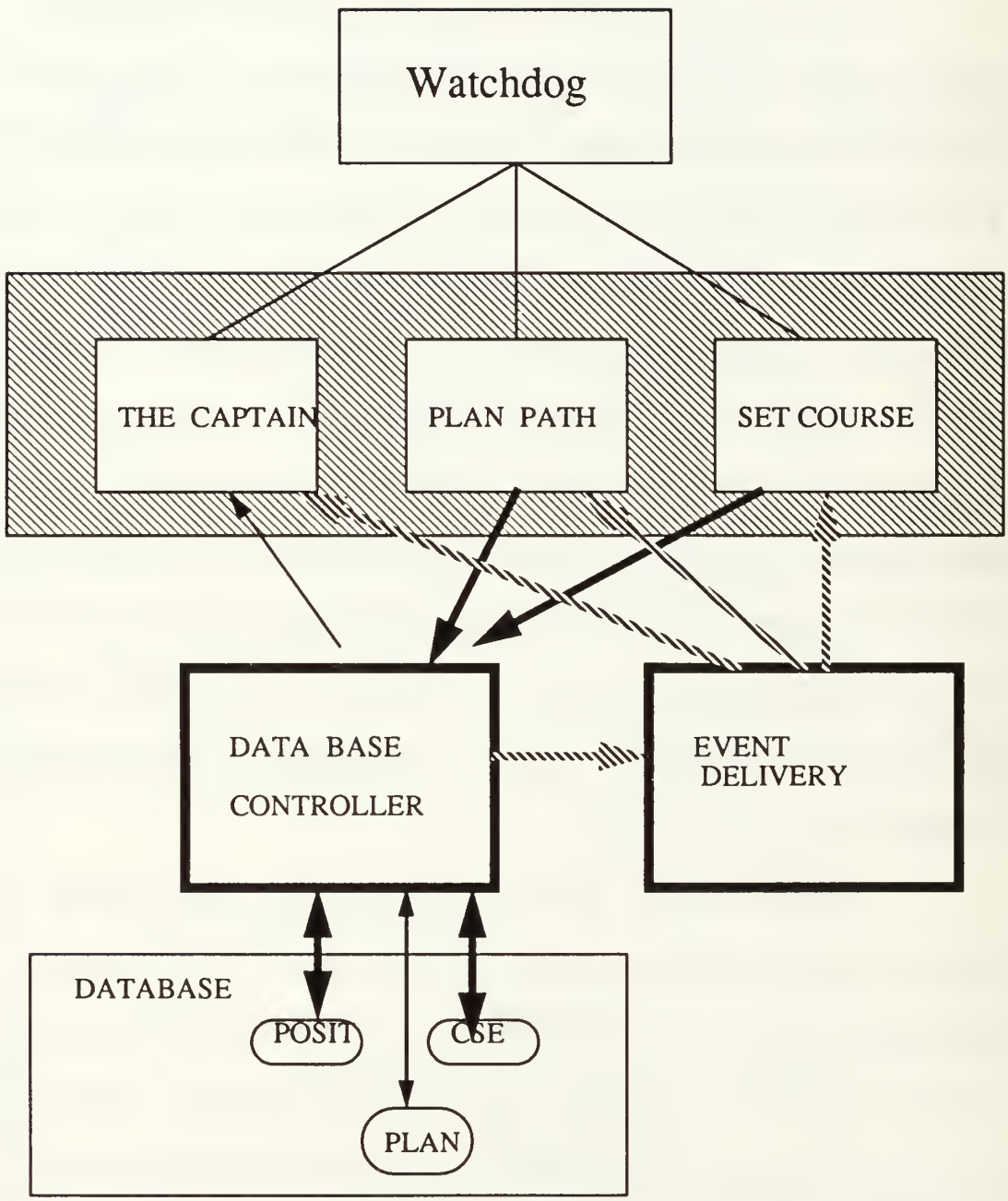


Figure 2-5. Module Interfaces in SACOR

implementation indicates that this software is not yet mature. Navigation is conducted by a dead-reckoning device rather than a combination of dead-reckoning and sonar comparisons as in the NPS AUV II. Speed, heading and depth can be controlled. A waypoint determination module allows for computation of the speed and heading to gain the next waypoint.

It is interesting to note that SINTEF project designers have noted for possible future work the development and integration of an intelligent captain which could reason about decisions and an intelligent watchdog for the vehicle internals (Rodseth 1990, p. 23). This is essentially the idea of a Mission Executor as outlined by the Naval Postgraduate School.

C. HYBRID MODELS

1. University of Karlsruhe Robot Project

Rembold and Levi have been directing research at the University of Karlsruhe, Germany into the control of autonomous vehicles with the 4-wheeled MOBILE ROBOT (Rembold 1986, pp. 79-80). They partition the control modules into a world processor, the planning and execution processor, and sensor processor. Rather than a pure vertical or horizontal hierarchy, Rembold and Levi describe their flow of execution as a hybrid of both. The real world model and the sensor module cooperate in providing the interpretation of sensory output and in storing the vehicle internal world. The planning and execution processor allows for comparison of a real-world scenario with the current

scenario to determine the action to be given to lower levels of guidance and control. The decision-making framework is a hierarchical, almost tree-like rule-based structure (Rembold 1986, pp. 80-83).

Levi and Rembold also require the software control system to do a limited amount of learning and to operate with incomplete information. MOBILE ROBOT must operate in an industrial environment and thus must be able not only to transit to the desired work area, but also to perform assembly tasks. (Because only the first mission is relevant to AUV at this point, only the transit execution will be covered.) The world model which MOBILE ROBOT depends upon has both static fixed obstacles and moving obstacles (Rembold 1986, pp. 81-84).

The vehicle planning and execution is carried out by a hierarchical control system very nearly like Isik's and Meystel's three-tier hierarchical control model. Command flow and generation are executed in the classic waterfall method. A global path plan and executor is responsible for the highest levels of decision-making and adaptation. An expert system at the highest level determines the route using a cube-based representation of free-space. The global path planner must transform parameters of decisions based on the overall route, obstacles or obstructions, and path constraints (percentage deviation allowed for various missions) into cartesian coordinates through an intermediate sequencer. This in turn passes the geometric coordinates to the Navigator expert system module which must control and interpret sensory output for navigation and recognition of various obstructions and provide adaptability strategies for local deviations to the path. Cartesian coordinates are translated to vehicle subsystem actions which are

in turn passed to the pilot level (which corresponds to the NPS guidance level). An expert system actually coordinates vehicle actions at this level to avoid contradictory guidance system actions (Rembold 1986, p. 85). The software architecture is shown in **Figure 2-6**.

2. Procedural Expert System (Esprit Project)

Procedural expert systems are the object of this cooperative research between the University of Amsterdam and Framentec of Paris on an industrial robot (Meijer 1990, p. 65). Essentially what has been constructed is a mission executor. Meijer and his colleagues have constructed a model known as the Exception Handling Model. A stack structure is used to store the current operations that the robot is performing. The operations that the robot can perform are classified according to complexity. As with any robotic application, planning and initial scheduling is conducted offboard the robot. Adaptive scheduling is generally required, as well as generation of recovery plans, to deal with any interruption to the preplanned operations (Meijer 1989, pp. 65-66).

The Exception Handling Model attempts to achieve the planned behavior and provides a series of prioritized strategies for recovery. Like many other robot models, it structures them in heuristics around the general functions of monitoring, diagnosis and response in a loose hierarchy. Fault trees are used in the diagnosis part to trace a component failure. Recovery plans are generated from this. Each possible strategy is checked for feasibility. The system will default to a rescheduling mechanism if recovery with the current goals is not possible (Meijer 1989, pp. 66-67).

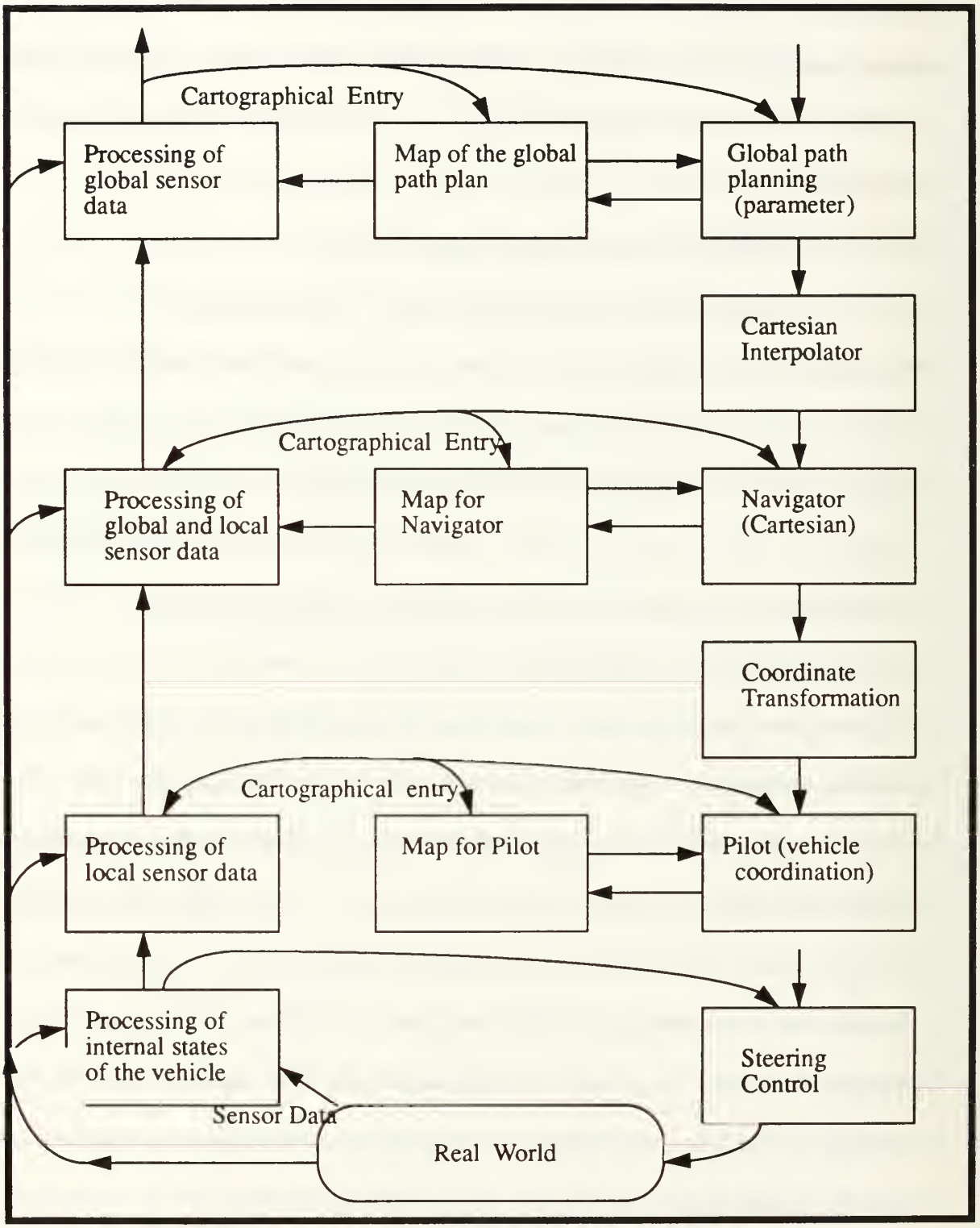


Figure 2-6 . Software Control in the Karlsruhe Robot

The Procedural Expert System itself is programmed in LISP and consists of a knowledge-base which contains the vehicle and environmental model states and the so-called *Knowledge Area.* These are essentially structures similar to rules which have prerequisite facts that make up an interface. These have associated with them some type of procedural code. This is the Procedural Expert System's method of encapsulating general plans and domain-specific plans. It is very nearly a paradigm of polymorphism. A structure similar to an inference engine selects the *Knowledge Area* to be executed depending on its facts being resident in the knowledge base. Goal-achieving *Knowledge Areas* can essentially invoke one another in a fashion similar to the classic forward-chaining mechanism of rule-based systems. Constraint-based backtracking is available to assist in truth maintenance for the knowledge-base (Meijer 1989, pp. 70-75).

Exception-handling is structured into knowledge areas specifically designed for that purpose. These invoke the regular task achieving knowledge areas (Meijer 1989, pp. 70-75). Although stack - *Knowledge Area* interaction is not explained in detail, there is mention of pursuing new goals should that become necessary. In that case, the next available goal would be removed from the stack for activation. Tasks have a hierarchical flavor, yet the underlying reasoning is not developed into a true hierarchical software architecture.

D. SUMMARY AND EVALUATION

This limited survey of AUV software architectures indicates that there is some conceptual agreement in architecture but wide division in implementation. Rule-based

systems are popular, but are implemented in different fashions. Subsumption is part of several architectures, yet it is not effected in the manner that Brooks originally devised. The TASC/NUSC model is most similar to NPS AUV II in terms of Mission Executor design. However, the TASC/NUSC model makes a further division for task execution managers which the NPS model does not. The TASC/NUSC model implies that there is an object or module to monitor each of the critical evolutions. SINTEF Corporation's object-oriented AUV model, SACOR, has similarities to the NPS AUV II, but it assumes a more distributed mission executor. Actually, there is no distinct module known as the executor in SACOR. Most of this functionality is derivable from the Watchdog and Captain modules. Unfortunately, the Captain module is merely a sequencer with no intelligence, heuristic or otherwise (although intelligence is planned for possible incorporation as the project matures).

The layered hierarchical control models, while presenting a non-traditional approach to robotics architecture, present a very credible method of testing software. While all researchers may not agree on Brooks's subsumption of behaviors model, the incremental addition and verification of the software is currently being carried out in AUV II. Division of decision-making and control in AUV II is evident in only two explicit layers. Implicitly, the navigation module, pattern recognition software, vehicle condition monitoring module, and guidance module are all in an intermediate level. Thus, one might be able to infer that the hierarchical models might be closest in design to AUV II. Most of these, however, have software redundancy in each layer as Isik and Meystel's Intelligent Mobile Autonomous System (IMAS) does. The NPS AUV II software

architecture would best fit in the hybrid category. It is not strictly hierarchical nor is it a layered control/subsumption model. Clearly, it is not the traditional horizontal model. The current implementation of the Mission Executor (as later explained) is situation-event based. Combining this aspect with the hierarchical structure, one must conclude that the NPS AUV II software is a new variety of the hybrid model.

III. THE C LANGUAGE INTEGRATED PRODUCTION SYSTEM

This chapter provides an overview of the C Language Integrated Production System (CLIPS) and the arguments for its use as the ongoing tool for construction and extension of the Mission Executor of the Autonomous Underwater Vehicle in both simulation and the actual testbed, AUV II.

A. MAIN FEATURES

CLIPS was developed to meet the need for a low-cost, portable, rapid prototyping tool which could be used in the construction of both real-time and non-real-time systems. The effort was begun in 1985 at the NASA Johnson Space Center with construction of a prototype. The intent of the design was for CLIPS to mimic features of both the Automated Reasoning Tool (ART), the List Processing (LISP) language and Official Production System 5 (OPS5). The Software Technology Branch at NASA was essentially successful in this venture. CLIPS version 3.0, the first to be released to users outside of NASA, was distributed in 1986. Since that time, the expert system has undergone several revisions. CLIPS is a forward-chaining, rule-based tool which, like the production systems it is based on, uses the Rete algorithm for pattern matching and inferencing. (NASA 1991, pp. xiii-xiv)

CLIPS rules are generally constructed of facts in the relation-attribute and associated value form on the left side of the production arrow (=>). The asserted facts which are produced are placed on the right-hand side. **Figure 3-1** depicts a sample defrule which

may be found in the Mission Executor system. Facts may have constraints placed on their values. Logic expressions such as conjunction, disjunction and negation (in the form of ***and***, ***or***, ***not***) may also be attached to them. CLIPS allows for efficient pattern-matching on variables on the left-hand side. Procedural statements such as If...then...else and while-loops are available. Truth maintenance is available through the use of the ***logical*** construct to assert a fact (or facts) which has a dependency. Retraction of one of the original left-hand side facts removes the support for that assertion. This is illustrated in **Figure 3-2.** A substantial numeric function library is available for logical comparison, some conversions of standard units to others (degrees to radians and vice-versa), and special numeric evaluations. CLIPS input and output (I/O) is very similar to LISP and the Common LISP Object System (CLOS). Formatted input and output is nearly identical to LISP. (NASA 1991, pp. 5-47)

Earlier versions of CLIPS did not include any object orientation or user-defined functions. User-defined functions had to be written in the source language. Version 5.0 now includes the CLIPS Object Oriented Language (COOL) which exhibits properties of both SmallTalk and CLOS, and user-defined functions. It supports a frame hierarchy of classes and objects. Presently it only supports specialization inheritance (although there is a way to emulate generalization). Like other object-oriented systems, CLIPS 5.0 provides inheritance and strict interfaces (message-handlers) for accessing the data in objects. Procedural constructs such as *daemons* may be attached to objects and fire upon basic actions such as initialization or modification of slots in an object. Impressive

```
(defrule    monitor-battery
   (action    monitor)
   (current-time ?time&:(> ?time ?*guardline*))
=>
 (assert (battery     time-critical))
 (assert ( guidance    shift-power-source)))
```

- This rule is typical of an automated control-type rule
- ?time is a constrained variable. This rule will only fire if ?time is greater than the global variable ?*guardline*, which must be elaborated at run time.
- On the right-hand side, two facts are asserted. The second one is typical of a control fact. It causes another module to execute another rule (semantically-linked).

Figure 3-1. A Sample CLIPS Rule

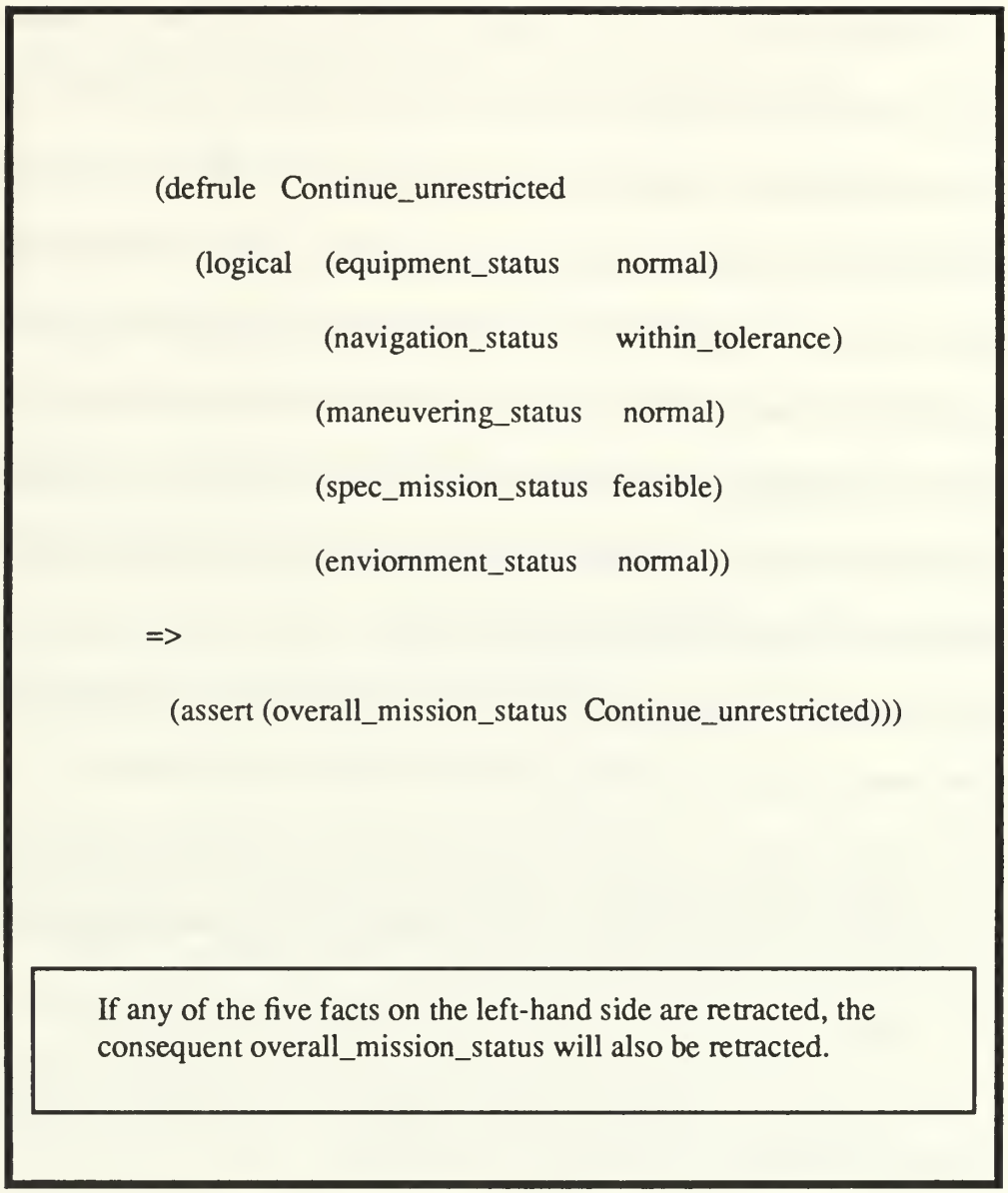

(defrule Continue_unrestricted
(logical (equipment_status normal)
(navigation_status within_tolerance)
(maneuvering_status normal)
(spec_mission_status feasible)
(enviornment_status normal))
=>
(assert (overall_mission_status Continue_unrestricted)))
If any of the five facts on the left-hand side are retracted, the consequent overall_mission_status will also be retracted.

Figure 3-2. Use of the CLIPS Truth Maintenance Construct *logical*

polymorphism allows even a casual programmer to create a *defmethod* which will operate differently when presented with arguments of different types. (NASA 1991, pp. 5-18)

Efficiency in CLIPS is due primarily to the use of the Rete Algorithm. A recent synthetic benchmark conducted by Mettrey at Bell-Northern Research demonstrated that systems using the Rete Algorithm were substantially more efficient and faster than their competitors which used a different pattern matching scheme. Writing conditions that specify a rule is instantiated only if a pattern cannot be matched by any fact in the knowledge base is a powerful feature of the Rete-based tools. Temporal redundancy, a common characteristic of knowledge-based systems, is used to great advantage by Rete-based tools. Rete saves repetitive information on nodes and propagates only changes, thus increasing efficiency. (Mettrey, 1991, pp. 19,30)

In addition to low cost, CLIPS has been designed with a great deal of flexibility. It has many features of more expensive tools, including the following:

- CLIPS does not require the entire environment to be available on the operating system to run an application. Executable modules can be created which allow econmomical delivery of the application. (Riley 1987, pp. 33-40)
- CLIPS is portable to any environment supporting a C compiler.
- Seven different conflict resolution strategies are available rather than just depth-first-search. (NASA 1991, pp. 28-31)

CLIPS's only apparent weakness is an absence of pattern-matching on the left-hand side for objects in CLIPS Object Oriented Language (COOL). Some NASA

programmers readily admit that this is an impediment in some applications. On the other hand, there are work-around solutions to this.

B. DEVELOPING CONTROL EXPERT SYSTEMS IN CLIPS

The need for a low-cost tool such as CLIPS is evident by its widespread use in the government, commercial, and academic communities and by the proliferation of software systems constructed in CLIPS since it was first released. A recent advisory released by the NASA-Johnson Space Center indicated that over 3000 users are programming in CLIPS (NASA 1991, p. xiv). The range of applications has included robot control expert systems, advisory systems, intelligent tutoring systems and numerous embedded applications. As this research is primarily directed at high-level control, a small sample of some of the control applications completed or in development follows.

Case Western Reserve University's Center for Automation and Intelligent Systems Research developed a model-based space station autonomous power control system in 1988. The simple model used, essentially a terrestrial one, requires the power control system to dynamically schedule many power loads for a station with but a single power source. Three phases of power control are modeled: a normal phase, an emergency phase, and the recovery phase. Heuristics are embedded in the rules which deal with predictions and consequences of possible load failure. The operator is warned of impending failure as the system moves through phases of warning, critical and failure. If the operator takes no action, the system will automatically shed the failing load.

Only seven basic supervisory rules are used to control the system. All use very basic CLIPS patterns and virtually no frame-based templates or complex objects. (Vezina 1988, pp. 211-220)

The Center for Engineering Systems and Research (CESAR) at Oak Ridge National Laboratory has implemented a robotic expert system in CLIPS version 4.0 which allows a robot to find and operate plant controls in a hostile atmosphere such as a smoke-laden control room. The object of this was to implement machine learning. (Spelt 1989, pp. 8-15)

Elcee Computek Incorporated has been developing a guidance system simulator known as KMARS (Knowledge/Geometry-based Mobile Autonomous Robot Simulator) for robotic vehicles. This includes both a knowledge base (written in CLIPS 4.3) and a geometry base. The simulator plans and executes motion for a point robot in a two-dimensional environment. The expert system calls C language functions to execute procedural activities. The expert system attempts to determine if a geographical goal can be located by a limited range sensor. If the goal cannot be "seen" by the sensor, a subgoal is created. When the point vehicle reaches the subgoal area, the vehicle sensors are again activated to see if the goal can be detected. The overall purpose in this system is to explore unknown environments with little a priori knowledge (Cheng 1990, pp. 822-830). This application is similar in nature to the general autonomous underwater vehicle problem and is one more indicator that CLIPS is a proper tool for this application.

C. COMPARISONS AND BENCHMARK

The recent virtual explosion of available expert system tools has made selection of the appropriate tool for an application a daunting task. William Mettrey of Bell-Northern Research recently compared five well-known tools [ART-IM, VAX OPS5, Level 5, KES] for adaptability and support of the these commonly desired characteristics:

- knowledge representation
- inference
- development environments
- delivery environments
- documentation
- support (Mettrey 1991, p. 19)

Mettrey found the inferencing capabilities of CLIPS to be very strong. The Rete algorithm upon which it is based is a very appealing and efficient algorithm. Despite this, Mettrey criticizes CLIPS for not having frame-base reasoning. (At the time of publishing, CLIPS version 5.0 with the CLIPS Object Oriented Language had not yet been released.) Further, the development environment is not as advanced as some of the other tools (Mettrey 1991, pp. 20-21). CLIPS 5.0 is currently being updated to include a more advanced development environment with a mouse-driven interface.

Naturally, there was a strong tendency to measure less esoteric facets of the development tools. Mettrey devised a synthetic benchmark that consisted of typical

rules, consisting of object-attribute-value facts (some with constraints) on the left side and fact assertions on the right side, which are commonly found in rule-based expert systems. Seven different cases were examined. In the first case, twenty-five typical rules were placed in the program. Timing began at run-time and ended at 250 rule-firings. The number of rules inserted and the rule-firing termination point were increased by a factor of two in each of the succeeding cases. Timing analysis was conducted on a Sun 3 workstation, a MacIntosh II, and a VAXstation 3100. Knowledge Engineering System (KES) and CLIPS were first compared on the Sun 3 workstation. CLIPS outperformed KES quite dramatically : a ratio of 12.7 to 1 in speed on the low-end case, and 19.5 to 1 in the high-end case of 200 rules with a termination point of 2000 rules. On the Macintosh II, CLIPS performance over Level 5 was less dramatic but still significant. VAX Official Production System 5 (OPS5) performance on the VAXstation 3100 was marginally better than CLIPS. CLIPS fired rules slightly faster than the Automated Reasoning Tool for Information Management (ART-IM). This is interesting inasmuch as CLIPS was designed around the characteristics of ART in its original form although NASA claims that no actual ART source code was used. Mettrey notes that Inference Corporation, which developed ART, later used CLIPS as the base for its development of ART-IM. (Mettrey 1991, pp. 22)

Although the benchmarks were useful in determining performance among the tools, a metric such as this is of limited value. Extensions on performance in all types of systems cannot be predicted on the basis of this evaluation. Theories of rule groupings have evolved which indicate that performance may be drastically changed by

the order in which rules are grouped in a rule-based program. One of the four expert system tools evaluated, Level 5, does not use the Rete algorithm.

Nonetheless, what can be observed from Mettrey's benchmark is that CLIPS, for its cost, is the best forward-chaining expert system tool among those evaluated. Further, version 5.0 (and its forthcoming subsequent version) has an object-oriented systems which is more tightly coupled than ART-IM, which lacks a few of the commonly recognized object-oriented features such as multiple inheritance.

D. PORTABILITY

As this is a specifically stated goal of initial CLIPS development, it is not surprising that portability is a notable strength. CLIPS can virtually be used in any environment which supports a standard C compiler. Mettrey's synthetic benchmark described above used CLIPS as the standard of comparison because it was the only tool which ported to all three versions of hardware previously described (Mettrey 1991, pp. 28).

CLIPS applications can be completed as compiled run-time modules in C or in the interpreted mode of the full CLIPS environment. As the environment is not large (currently less than 1 megabyte) and the speed-up of the compiled version only slight, in many applications it may not be to much advantage to convert to a compiled version except to save memory.

Further supporting wide portability is the fact that CLIPS comes with its source code. It thus can be customized for virtually any application. The CLIPS Advanced

Programming Guide gives explicit instructions for creating run-time modules and embedding CLIPS in applications in which the main program is written in C, Ada or FORTRAN. Run-time modules are created by first compiling the CLIPS source, loading all files of an application to the CLIPS environment, and then using a command known as constructs-to-c to convert the total application program to a series of C files. After modifying the header files, the CLIPS source main program is modified and the CLIPS modules linked together. The run-time modules are not suitable for an application which has the build/eval functions (NASA, 1991b, pp. 99-104). Thus, if an Artificial Neural System is to be simulated, it must be achieved through dynamic salience only.

Embedding an application requires a similar approach. CLIPS user-defined functions may be called via the CLIPS Function Call. Constructing objects requires the CLIPS Make-instance call in the source language. After the Load Constructs command is given for all of the CLIPS functions, the newly created C files are linked (NASA, 1991b, pp. 35-98). Porting an embedded application, like a run-time module, is relatively simple.

The GESPAC MPU30HF with Motorola 68030 CPU currently used in the AUV is well-suited to handle C-based tools. Thus, porting the Mission Executor should not be a monumental task. The current vehicle software is ported via RS232 interface. The OS-9 operating system is designed as a multi-processing environment and thus can easily support CLIPS.

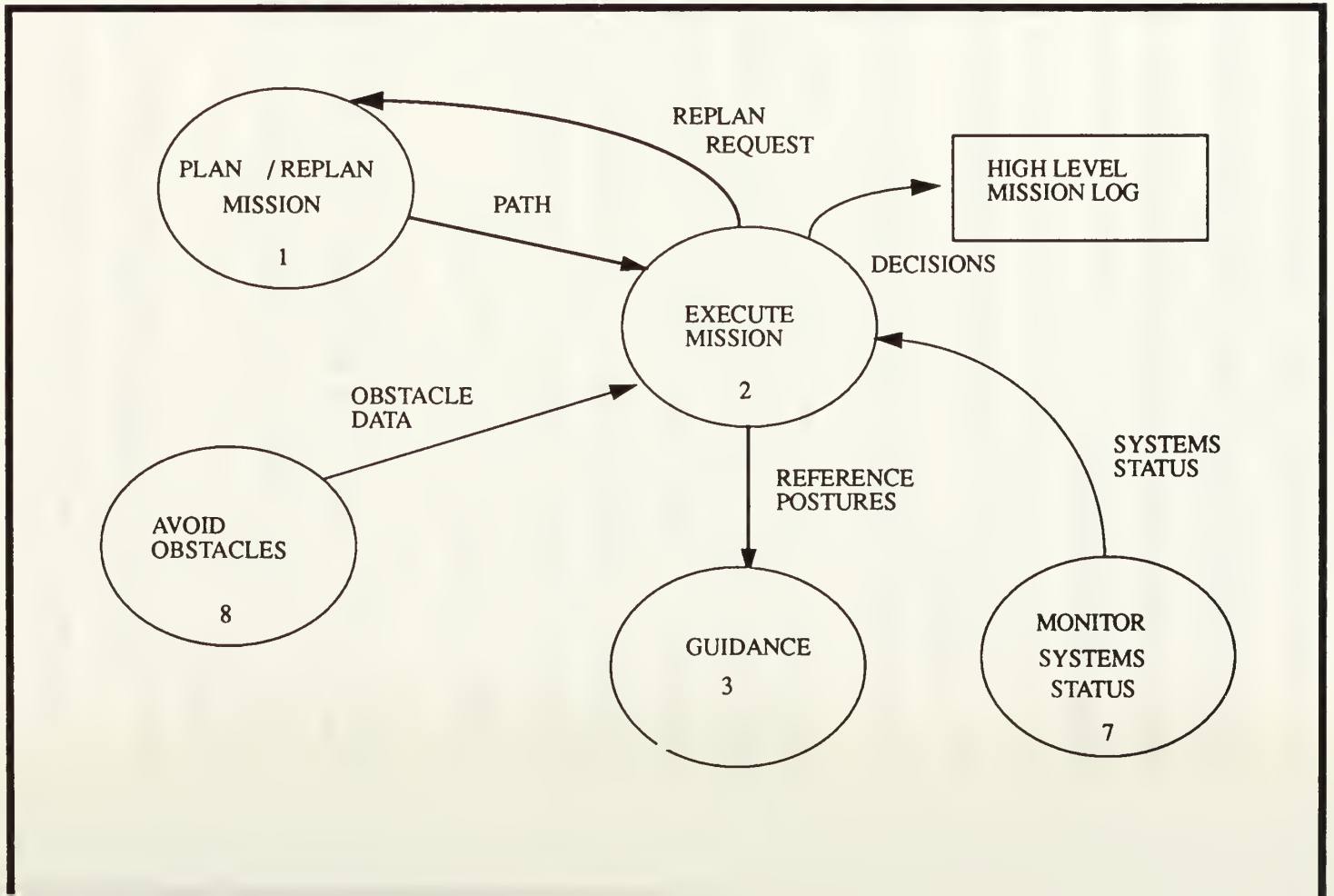
IV. ONBOARD INFORMATION PROCESSING

This chapter examines the data flow between the Mission Executor and other modules. The Mission Executor receives its path constraints and baseline commands from the proposed interfaces to cooperating modules (depicted in **Figure 4-1**) are discussed.

A. DOWNLOADING POSTURES AND COMMANDS FROM THE MISSION PLANNER

The offboard Mission Planner was successfully implemented by Ong (Ong 1990) and is being extended by Caddell (Caddell 1991). It provides a best three-dimensional path-to-goal given chart features of the region in which it is to operate, time requirements, and special path constraints. The Mission Executor's most important functions in a normal transit are to receive waypoint and command data (denoted as a *path*) from the Planner, interpret the movements, convert the path postures to reference postures, and properly sequence the movement. The path data are passed to the Executor in a file. The Executor converts the plan to a series of waypoint objects and then begins the monitoring of these objects. The other functions which the Executor carries out, while important, are generally exception-handling relative to normal operations.

This is not to categorize the Mission Executor's interplay with the Planner as simply one of a conversion unit serving a high-level planner. The Mission Executor


PLAN / REPLAN
MISSION
1
REPLAN
REQUEST
PATH
HIGH LEVEL
MISSION LOG
DECISIONS
EXECUTE
MISSION
2
OBSTACLE
DATA
AVOID
OBSTACLES
8
REFERENCE
POSTURES
SYSTEMS
STATUS
GUIDANCE
3
MONITOR
SYSTEMS
STATUS
7

must reason about these waypoints and the associated speeds. If the original commanded speed for a particular waypoint is no longer valid due to an unplanned deviation from course, then the Executor must call the Navigation module for an updated speed to get to the goal on time.

B. UPDATING FROM THE OBSTACLE AVOIDANCE DECISION MAKER

Conceptually, the Obstacle Avoidance Decision Maker has the responsibility for processing packaged sonar data from the pattern recognition module and relating it to specific obstacles. Decisions on both the type of obstacle (moving or stationary) and the avoidance maneuver (decrease-speed, increase-speed, dive, ascend) are determined and passed to the Mission Executor. One proposal for the manner in which it will pass data is an obstacle alert-and-direction flag followed by a template of the form:

- obstacle identification
- relative distance
- relative orientation
- time
- movement
- parameters of movement

The direction flag is sent merely to alert the Executor to a real-time report. Receipt of the template data allows the Executor to call the RePlanner with the information while also flagging Guidance to be ready for imminent receipt of new

reference postures for the new path-to-goal referenced to a new origin (the current geographical position). A low-level reflexive response can also be passed directly to the guidance controller bypassing the Mission Executor in the case of an unplanned obstacle close-aboard (Healy 1990).

C. UPDATING FROM THE SONAR MODULE

The Mission Executor normally depends upon obstacle identification and orientation data passed from the Obstacle Avoidance Decision Maker. Thus, sonar data from the pattern recognition module is filtered through the Obstacle Avoidance Decision Maker. Currently however, this is only a conceptual framework as the Obstacle Avoidance Decision Maker has not yet been fully realized. To bridge this temporary software gap, a proposal by Floyd to pass a four-bit flag directly from the pattern recognition module has been implemented (Floyd 1991). Depending on the pattern received, the Mission Executor will opt for a right turn, a left turn, an ascent, or any combination of these for gross avoidance. It will then request a new route plan from the RePlanner if there is sufficient need. Consideration of all features of an object, as in the template described above, cannot be achieved in this configuration without the intelligence provided by the Obstacle Avoidance Decisionmaker.

Therefore, the granularity to determine if an obstacle requires a *significant deviation* from the original track such that a new route must be planned becomes quite coarse. The Mission Executor takes this into account when performing the so-called "sensibility check" when the RePlanner provides a new route. The presence of any

obstacles on the new initial leg is quickly checked. More importantly, however, the current gross vehicle energy state is balanced against the distance-to-go along the new route. Nonetheless, due to the weakness of this method without the intervention of an Obstacle Avoidance Decision Maker, there may be several crossover situations in which a small deviation from the original path may unnecessarily cause a new route to be planned. This is not cause for concern in the AUV II's testing environment at the NPS pool because the turns are 90 degrees by default. Further, the pattern recognition software has the ability to disregard obstacle features which may be distorted while changing heading, thus avoiding an even great error in maintaining the desired path (Floyd 1991).

D. INTERFACE WITH THE REPLANNER

The RePlanner, a knowledge-based path-planner which uses an optimized real-time A* search, attempts to plan a new path-to-goal based on knowledge of the goal state, the current geographical location and special path constraints passed by the Executor. It operates in four dimensions: three standard cartesian dimensions and a fourth dimension of heading or azimuth (Bonsignore 1991). The RePlanner receives periodic updates from the environmental database, allowing it to replan the new route from any specified origin.

The RePlanner is alerted to the need to replan by a function call from the Executor. A flag and the coordinates of the current location are transferred to the

RePlanner. It constructs a new plan in the same manner as the Planner, using a priori knowledge of the environment. A file of new waypoints is returned to the Executor.

E. UPDATING FROM THE VEHICLE CONDITION MONITOR

Currently, the Vehicle Condition Monitor is not modeled at the real-time level. The vehicle's internal world is modeled as a set of sensor objects which measure the subsystem components. Objects are instantiated for power sources such as the array of batteries for subsystem power and propulsion support, control system indicators for rudders, planes and propellers, sonar power status indicators for the four onboard sonars, onboard computer temperature sensors, navigation instrument fault sensors, and power sources for environmental sensors. These have default guard-line and red-line ranges which, when violated, cause an alarm to be sent to the decision-making levels. An automated turn-key operation is first generated which attempts to balance an equipment failure or impending failure by bringing a redundant system on-line, if such redundancy has been provided. If the equipment is critical, it may degrade the mission status to continue-mission-restricted or to abort-mission.

The data interface must conform to strict object interfaces, as the subsystem sensors are modeled as objects. Each object is queried by its own appropriate message sent at regular intervals from the Executor. A message-handler checks the subsystem sensor object's slots to see if an operating parameter such as a temperature or power level falls within the guardline range. If it does not, the appropriate response is

generated. This may initiate the turnkey operation or may just cause the Executor decision makers to be notified. The object hierarchy is pictorially described in **Figure 4-2.**

F. INTERFACE WITH THE GUIDANCE SUBSYSTEM

The end result of the Mission Executor functions must be a series of reference postures and commands to the Guidance subsystem. Guidance is an intermediate-level function which has an algorithmic reasoning system within it. It converts high-level decisions and reference postures to low-level commanded postures for the Autopilot module. A function call within the rules of the Mission Executor generates an alert to the Guidance module to prepare for receipt of data and commands. The reference posture is modeled as an object and so passed to the Guidance module. Commands from the Executor to Guidance are sent as flags.

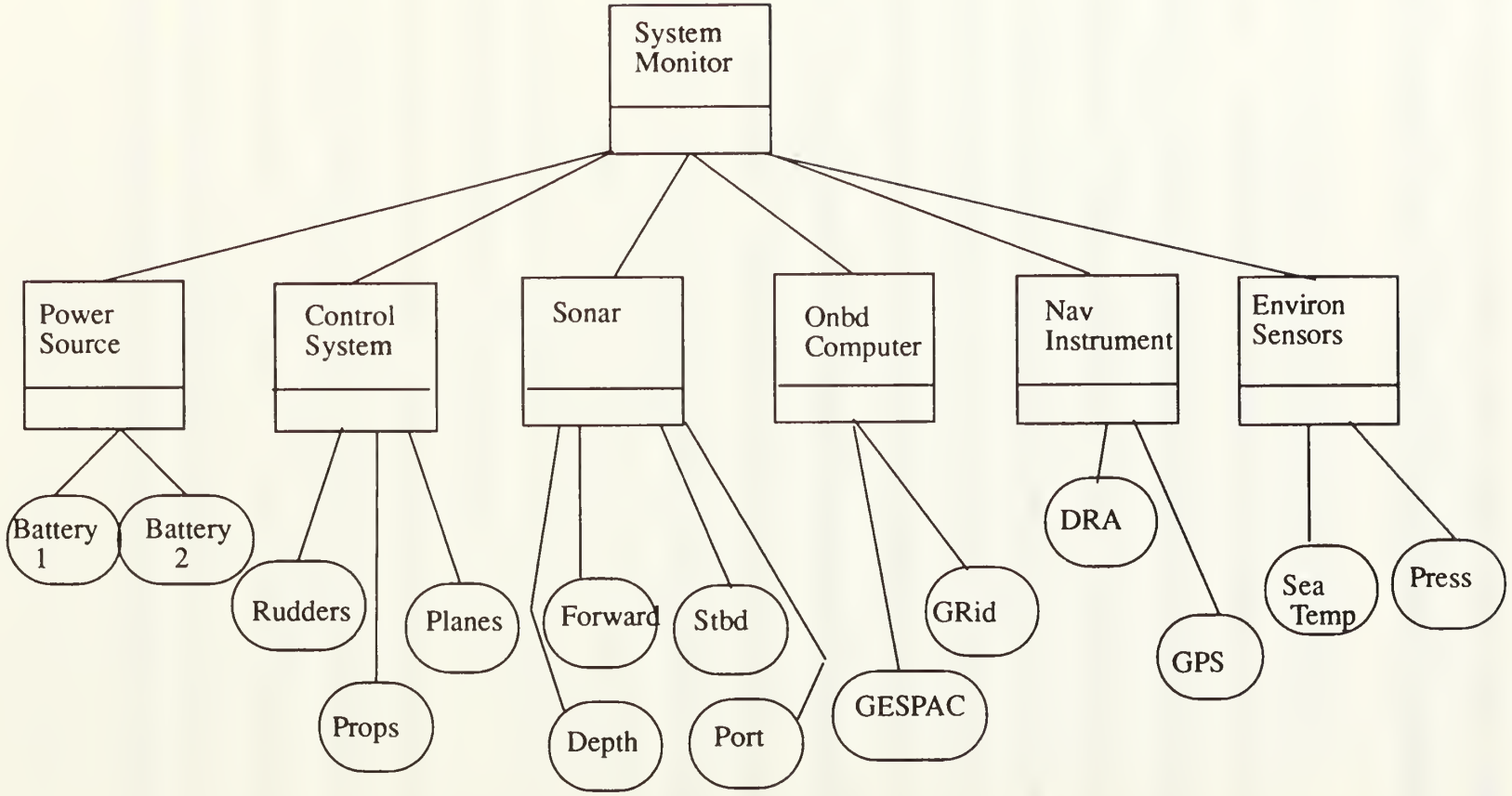


Figure 4-2. System Monitor Objects

V. DESIGN OF THE PROTOTYPE EXPERT SYSTEM

This chapter explores the design of the prototype, interface limitations, and the justifications for use of several of the software constructs. This design is intended to cover most AUV situations, but the current implementation is not considered in any fashion to be fully comprehensive.

A. PHILOSOPHY OF DESIGN: REASONING ABOUT SEVERAL WORLDS

The current NPS AUV II architecture is the result of an incremental development which began in 1988 at the conclusion of research for AUV I. Evolutionary changes in subsequent software design resulted in the need for a high-level control module. The Mission Executor, SKIPPER, attempts to fill the role of high-level director while integrating decisions based on input from three worlds: the vehicle's internal systems, the external environment, and the mission itself.

Simply put, the Mission Executor operates on more than one layer of symbolic reasoning. Decisions are modeled heuristically rather than in a strictly algorithmic fashion. High level decisions require a knowledge of the status of low level items to get a "sense of the system" and assess whether a mission can be carried out, which is the ultimate goal. The low level events then drive the broader decisions. The requirement to model this lends itself naturally to a hierarchical design, but one that is priority-situation based. Most AUV guidance/control systems are closed-loop and are equipped to deal with routine maneuvering. The Executor exists mainly to deal with

exceptions to normal maneuvering which cannot be dealt in a strictly algorithmic fashion. Its reasoning results in interrupt commands to guidance which controls the autopilot. If there are no deviations from the track caused by any of the three worlds that AUV must deal with, then the Executor merely fulfills a role of sequencer of data. The current implementation allows for the interface of system monitor functions which often are found on lower levels in other systems. However, as the current AUV II architecture does not charge the lower levels with this responsibility, both the intermediate and high level monitoring tasks are delegated to the Executor for the present. (This is expected to be replaced by an intermediate level module which responds to analog-to-digital outputs.)

Although not all experiential knowledge may be encoded in rules, there is reason to believe AUV missions can be bounded, at least for the time being. Some previous research has suggested that AUV behaviors might in fact be standardized. The University of New Hampshire's Marine Systems Engineering Laboratory (MSEL) and the Naval Underwater Systems Center (NUSC) cooperated in the research of some standard situations in which an AUV might find itself. The resulting matrix entitled "Generalized Problem vs Contingency Alternatives Matrix" they derived is interesting for its philosophy. Situations are classified in three categories of *problems*: mission, environment, and internal failures. These have a one-to-one correspondence with the three worlds that the NPS Mission Executor is trying to model at a high level. The authors, Westneat of MSEL and Clearwater of NUSC, determined that the AUV control system must be able of some limited decision-making for a (relatively) short

mission. Longer missions will require some form of machine learning which will not necessarily involve a neural net. (Westneat 1991, pp. 29-33) **Figure 5-1** shows a facsimile of the NUSC matrix.

This view of high level control as essentially handling exceptions to normal transit and operations is embodied in the Mission Executor. Some of the implications of the matrix merit serious consideration while others are simply beyond the scope of current technology. Vehicle self-repair is highly unlikely in a mechanical failure situation unless this term refers only to equipment which has a redundant system or power source available.

To implement the design described shortly, a number of assumptions about external modules are made. As some external modules remain to be completed, external module interfaces such as those described in Chapter IV are modeled as data files supervised by control rules. Scenarios are implemented by instantiated data from the files much as the expected module would perform. Files exist to model a module operating in two modes: (1) supplying data driven by demand from the Executor and (2) supplying data driven by events. Examples of type one are a command from the Executor to the Navigator module to provide the current location or a command to the RePlanner to provide a new list of waypoint postures. Event-driven data are inputs such as the initial list of waypoints from the offboard Mission Planner, obstacle data, and navigation reports such as waypoint data. This is depicted in **Figure 5-2.**

GENERALIZED PROBLEM vs CONTINGENCY ALTERNATIVES MATRIX

PROBLEM \ ALTERNATIVE	EMERGENCY ACTION	CHECK SENSOR	CHECK PLAN	WAIT FOR IMPROV	ANALYZE FOR CAUSE	CHANGE COEFF OF MODEL	REPAIR	ACTIVATE OTHER SUBSYSTEM	ORDER CHANGE IN RATE	CHANGE EVOLUTION SEQUENCE	REPORT PROBLEM	CHANGE PLAN OR GOAL SET
BECOME THREATENED	X			X				X		X	X	X
DETECT CHANGE IN THREAT BEHAVIOR	X	X		X		X		X		X	X	X
FALL BEHIND PIM					X	X			X	X	X	X
RECEIVE MISSION CHG										X		X
OUTSIDE INTERFERENC	X			X		X		X	X	X	X	X
PRIORITIES FOR GOAL SET CHANGE	X				X					X	X	X
DETECT ENVIRONMEnT CHANGE		X	X	X		X		X		X		X
SENSOR / MONITOR FAILS/	X			X	X		X	X		X		X
DETECT SUBSYSTEM FAILURE		X			X	X	X	X		X	X	X
PLATFORM FAILS/ DEGRADES	X	X			X	X	X		X	X	X	X

Figure 5-1. Naval Underwater Systems Center Matrix

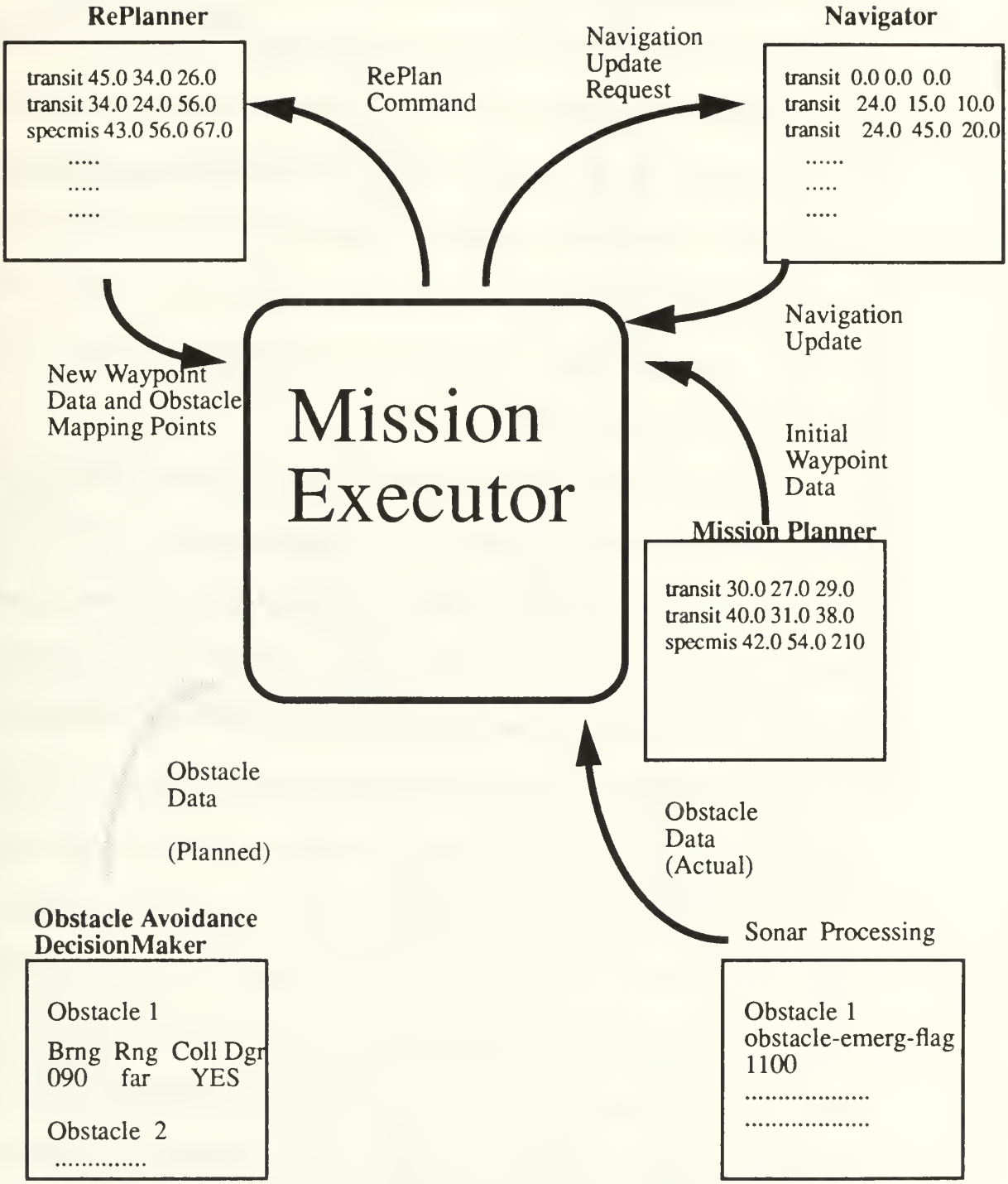


Figure 5-2. Event and Demand Driven Data

The basic overall design consists of a knowledge base of rules, facts, and objects. This knowledge base, although currently stand-alone, is expected to interact with modules such as the Obstacle Avoidance Decision Maker, and call external modules such as RePlanner and Guidance. **Figure 5-3** describes in a simple graphical fashion the overall schema for the Executor: a base of rules exists for each functional (i.e., situational) area: maneuvering, navigation, subsystem-monitoring, environmental, and specialized mission. The rules interact with the object base and cache of facts to produce the required guidance commands. Several global variables are used to represent performance parameters.

The rule base is instituted in a hierarchical fashion. The Overall Mission Assessor tabulates the status of each functional area. If no deviations occur during the course of the mission, the mission status remains at its default status, continue_unrestricted. It views each area in two levels: critical and failure. The critical level indicates that the functional area has suffered some sort of restrictive, non-catastrophic loss of capability. This can be on the order of loss of non-mission essential equipment or a temporary maneuvering restriction such as a obstacle avoidance which takes it from its principal direction of travel. This results in a mission status of *continue with restrictions.* The failure level indicates that the functional area has suffered a major loss of capability such as loss of mission-essential equipment or inability to maneuver. This essentially results in a mission status of *mission abort.* The mission restriction category can later be lifted if the vehicle recovers in ample time. If not, the mission restriction remains or worsens the overall mission status to *mission abort.*

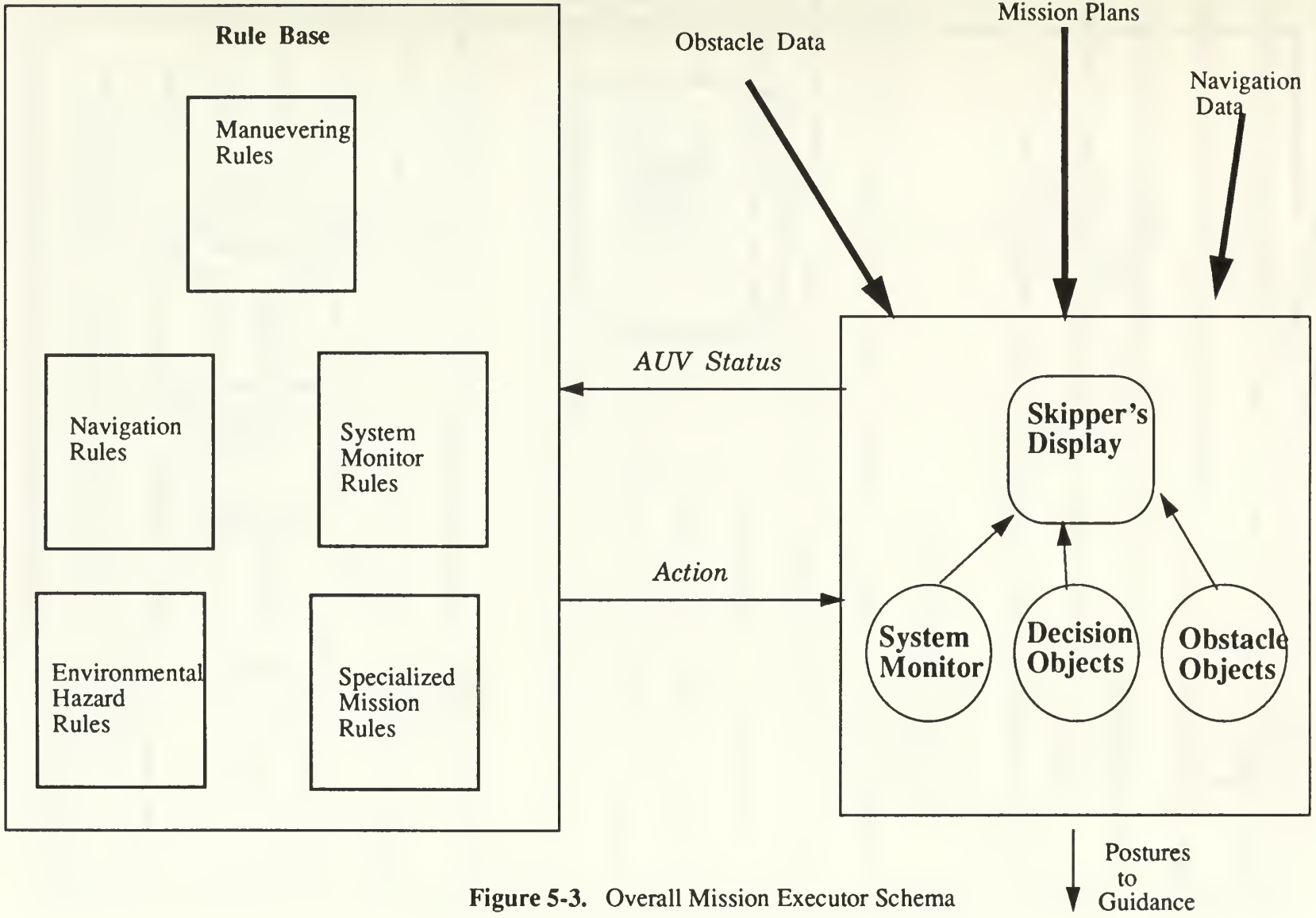


Figure 5-3. Overall Mission Executor Schema

The functional rule areas also have a hierarchy in themselves. A functional assessor exists at the top of each rule base to cache knowledge about the functional area. This then passes the functional area information to the main fact base which causes the executive decision rules to be fired. (The distinction between main fact base and functional area fact base is merely conceptual as the CLIPS inference engine does not perform this discrimination.) A schematic of this is shown in **Figure 5-4**.

B. SEQUENCE OF CONTROL

The sequence of control in a rule-based system often contains a relatively high degree of non-determinism because of its declarative nature. While there are certain tasks which must be accomplished in procedural order, as mentioned before the Executor is a system which reasons about situations which are normally beyond a closed-loop control system. The CLIPS inference engine does a depth-first search of a fact-node hierarchy, but the actual implementation hierarchy traversal is not quite as clear.

Input mission postures are first uploaded from the Mission Planner offboard the vehicle. As at the time of this writing not all AUV II software modules are implemented, the current version of the Executor assumes that a simple data file structure exists as the interface between the Mission Planner and the Executor. The input postures read from the file are given to a Mission Interpreter which places a posture into the proper object format and designates the high-level classification of the posture configuration as a transit or specialized mission. As the lower level

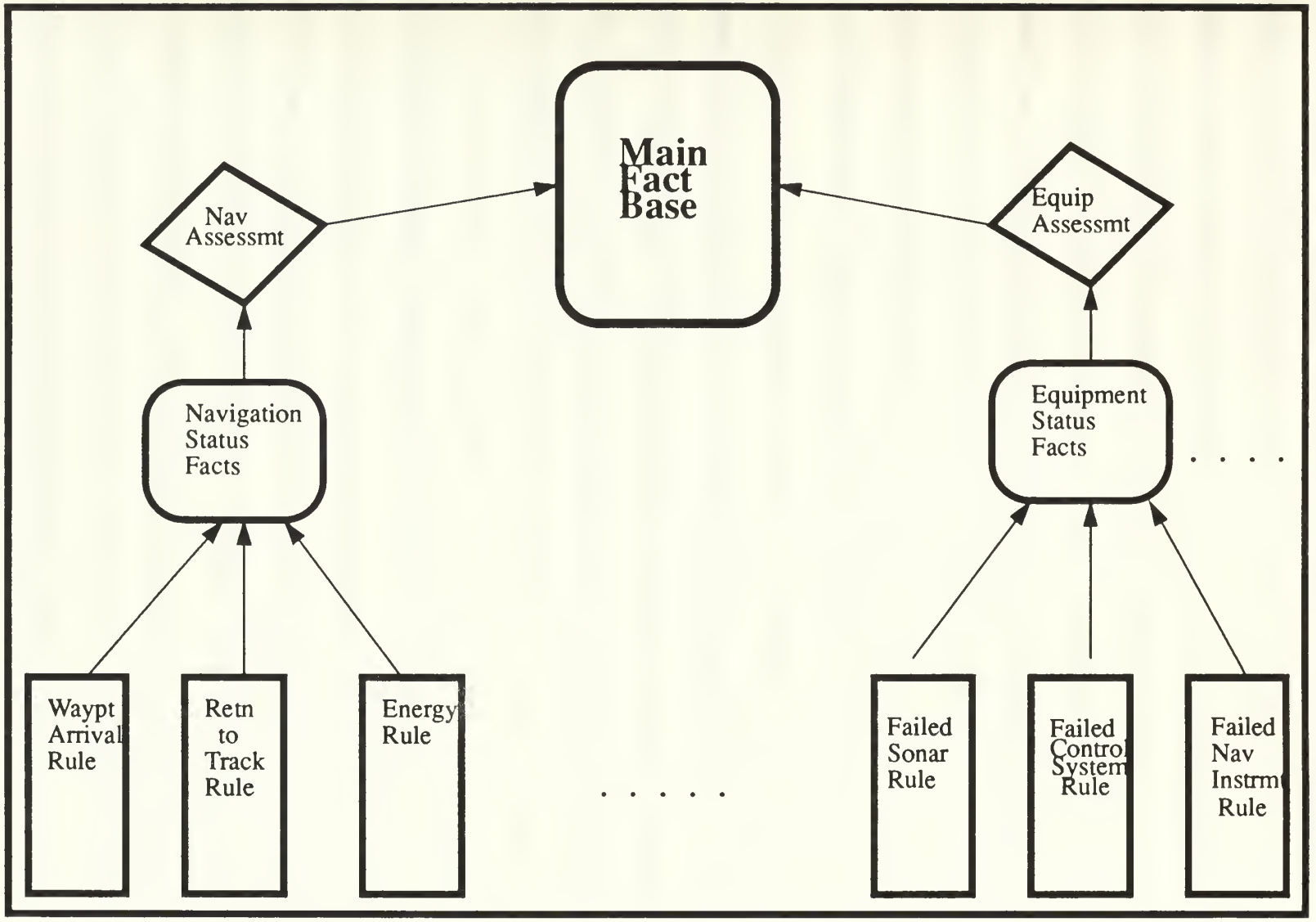


Figure 5-4. Functional Area Hierarchy

configuration of the posture (depth-level turn, ascent, dive) is unknown at that time, a comparison is made from waypoint to waypoint and the lower level action instantiated.

Not only a large influence for its own functional area, the Equipment Status area (or interchangeably Subsystem Monitoring Area) exerts a notable influence in other areas. Separate rules exist for each equipment area (sonar, control system, navigation instrument, environmental sensor and special mission equipment) and the respective power source. A continuous monitoring rule polls each equipment area for equipments which are out of out of normal operating limits. These limits are normally parameters of sustenance such as potential in volts or power in watts. If a mission essential equipment fails, it causes a failure in both the Equipment Status area and in the area with which it is associated. For example, loss of the diving-plane controls causes a maneuvering loss and a mission essential equipment loss. If an auxiliary power source exists for an equipment, it can be used in the event that the normal source fails. Similarly, equipment with redundancy has the capability to have its functions shifted to the alternate should it fail.

An Equipment Status Assessor awaits the results of equipment polling. If an equipment fails, then the equipment (previously classified as mission essential or not mission essential) will cause its equipment classification rule to fire and the Status Assessor will tabulate the results. If a mission-essential equipment or a sufficient quantity of non-mission-essential equipment fails, the equipment status area will suffer a major failure.

The instantiation of the lower level **action** attribute of the configuration actually takes place within the Navigation rules upon the occasion of waypoint arrival. Another rule which plays a large part in the navigational aspect of high level control is the assessment of progress along the mission track. The rule does a simple comparison of overall distance along the track with current location. It then orders a replan of the current track if the current speed and progress made are not compatible with reaching the goal area on time. A very simple energy-consideration function checks whether there is sufficient propulsive power to get to the goal.

Other navigation rules cover specially-monitored depths: both yellow depths and red depths. If the depth sonar indicates that the AUV has encountered a yellow depth area, AUV calls the Navigator for a check of the required depth in that area. If the observed depth does not match the required depth, guidance is ordered to reverse course and the replanner is called. If a red-depth violation is indicated, guidance is called to reverse course.

Maneuvering rules cover several areas. First and foremost are the obstacle avoidance rules. The highest priority rules cover emergency situations such as detection of an obstacle close aboard. The various orientations of the obstacle relative to the AUV's heading will prompt a right or left turn, an ascent or a full stop (drive motors stopped) or a combination of these. These are heuristic turning rules which proposed by Floyd which can produce an effective gross avoidance for the AUV so that the RePlanner can then be invoked for further path refinement (Floyd, 1991). Floyd's table upon which the Executor rules are based is reproduced in **Table 5-1**. This is essentially

TABLE 5-1. AUV OBSTACLE AVOIDANCE MANEUVERS
(Floyd 1991)

Obstacle Alert Flag Fwd,right,left,bottom	**Turn**	**Depth Change**
0XX0	----	----
0XX1	----	ascend
1101	left	ascend
1100	left	----
1011	right	ascend
1010	right	----
111x	stop	(ascend)

0 = No Obstacle **1= Obstacle** **X= 0 or 1**

an interim measure which will be replaced by a more detailed avoidance procedure in the forthcoming Obstacle Avoidance Decision Maker module.

Detection of an obstacle at the range of the sonar's limits is another function covered by maneuvering rules of the Mission Executor. Because of the AUV sonar's relatively limited distance, avoidance action must be taken early. The obstacle is initially checked for its potential to hazard the AUV. This is dependent on the obstacle's bearing drift and its relative bearing. This is recorded and a collective obstacle heuristic is instantiated to determine whether a proportional amount of obstacles will block the AUV to the left or right. A gross avoidance maneuver is then commanded to bring AUV away from the obstacle and allow the RePlanner to plan the new avoidance path with appropriate mapping waypoints.

The procedures for an update to an obstacle are essentially the same. If the obstacle is still a hazard, then further avoidance and replanning are necessary. There is a danger that this will result in a significant deviation from the path and that this will result in a mission abort. This is accounted for in the functional area assessment rule. If an obstacle is no longer a danger, then its collision danger is recorded as such and thus it is not considered in the collective obstacle assessment.

Other rules in the maneuvering functional area cover special depth-changing evolutions such as diving, ascents, and surfacing. The control systems have an inherently large influence on these special maneuvers. If a control system fails during one of these situation, that results in an automatically commanded maneuver to guidance to correct the attitude and level the vessel at a safe depth or change the speed

at which the maneuver is proceeding. An improper obstacle clearance can also precipitate changing one of these special evolutions.

The Environment rules have a similar arrangement. An Environmental Assessor tabulates the number of sensors which have performance readings which are out of limits. If it is an essential equipment such as the pressure transducer, the loss will cause a functional area loss. If it is a non-mission essential equipment, the loss will only cause a minor degradation to the environment functional area.

While basic AUV maneuvering control and navigation will be the primary focus for some time, incorporation of specialized missions will eventually become important. Specialized Mission rules have a different influence than the previous functional areas. Most of these rules do not take effect until the transition to a special mission configuration at the conclusion of the transit. The exception to this is a special mission area equipment failure. A functional mission area failure occurs if the special mission equipment fails. Future versions will most likely have an alternative to undertake a secondary mission if the primary mission cannot be fulfilled. The mission area rules, although not implemented in the current version , will probably be based loosely on MacPherson's description of AUV missions in template form (MacPherson, 1988, pp. 59-75).

As mentioned previously, the functional area assessors report to the overall mission assessor. This is located in a block of rules known as the Mission Executive, which constitutes the highest level of reasoning in the Executor. The overall mission assessor is insulated from details of the reports by the functional area supervisors. It

only remains for the overall assessor to tabulate the results. If complete failure in any area other than the environment functional area occurs, a *mission abort* results. Less than a complete failure in a functional area may cause a degradation to *continue with restrictions*. A mission degradation results in a phenomenon known as *status lock*. A mission status of *mission abort* results in the two other status rules being removed. Thus, even a seeming recovery cannot override a *mission abort*. A degradation to *continue with restrictions* can improve to *continue unrestricted* if recovery occurs in the mandated time frame.

Mission abort causes the vehicle path to be replanned for a pre-planned rendezvous point. It may be the origin of the mission or an intermediate point which facilitates recovery by the launching platform. Continue with restrictions allows the vehicle to try to recover from its maneuvering, navigation, or equipment restriction. In the future, it may also allow for altering of the mission.

Certain high-level behaviors are modeled using the Artificial Neural Paradigm suggested by Giarratano (Giarratano 1991, pp. 228-229). This application of the salience of a rule is useful in differentiating between a high-level, less frequent action and a lower-level frequently performed action. The philosophy for using salience in this manner is that a situation (pattern match) which may cause a mission-abort or mission-restriction usually requires immediate or timely reaction and certainly takes precedence over a routine action such as a normal turn or depth-change in a normal deep-water environment. The emergency-action rule must be fired before other semantically lower-priority rules on the agenda. This (however loosely) heuristically

models a submarine commander's "situational awareness" in an emergency. It might also be likened to a focus of attention approach, such as that modeled by Blidberg and his associates at the Marine Systems Engineering Laboratory (Blidberg 1990, pp. 40-41). **Figure 5-5** illustrates an example of this.

Salience is also used in some background functions such as the sequencing of the mission timer and the continuous loop which queries the slots of the system monitors. Still, it is used sparingly. SKIPPER still retains a strong declarative nature.

The Mission Executor must send not only reference postures to the Guidance module, but commands as well. Many of the commands must initiate time-constrained lower-level actions while the assessment of a particular functional area status is in progress. The commands must be a series of well-understood actions which will place the vehicle in a safe configuration when a casualty occurs. The table of these commands is shown in **Table 5-2**.

C. TRUTH MAINTENANCE AND THE ROLE OF UNCERTAINTY

1. Maintaining a Consistent Knowledge Base

As important as sensing data and scheduling actions based on it is the maintenance of consistency in the knowledge base. In a rule-based system this becomes acutely important when the generation of a new action through a control fact is based on some other events. If the events which would cause that action are no longer valid, then it *may* be the case that the generated control fact is no longer valid. In such a case it would be necessary to go and remove the fact. This can involve complex rules. It

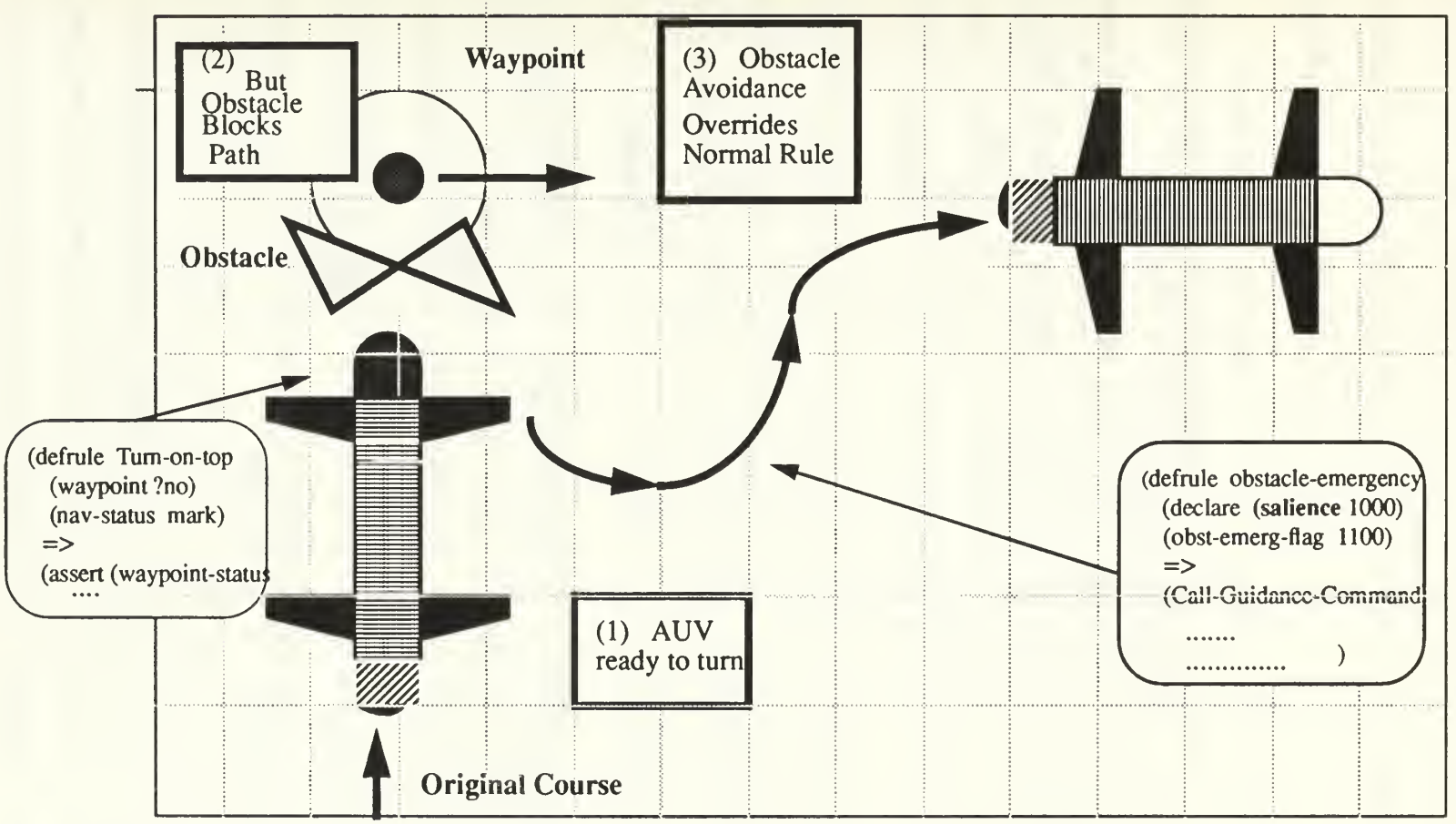


Figure 5-5. Situational Awareness Through Salience

TABLE 5-2. Executor Commands to Guidance

Basic Maneuver	Order	Object of Order
TURN	turn-left	rudder
TURN	turn-right	rudder
DEPTH-CHANGE	ascend-XX	planes
DEPTH-CHANGE	dive-XX	planes
DEPTH-CHANGE	surface	planes
SPEED-CHANGE	Increase-Speed	drive-motors
SPEED-CHANGE	Decrease-Speed	drive-motors
SPEED-CHANGE	STOP	drive-motors
SPEED-CHANGE	HOVER	hover-thrusters

XX= depth in inches or an indicated safe depth variable

can also be achieved through the use of the CLIPS *logical* construct discussed previously in Chapter III. One can withdraw a fact which is no longer consistent and is no longer supported (NASA, 1991).

Nonetheless, there are occasions when the logical construct is not as useful. These situations usually require some sort of search. Certain high level decisions may require knowledge of previous decisions. This is particularly true for the high level mission decisions. A previous instantiation of *abort mission* cannot allow for improvement to a better status as the abort mission should only take place when all relevant options to continue the mission have been explored and found insurmountable. The status lock feature helps to maintain the high-level configuration while still allowing for the necessary actions of avoiding obstacles and performing routine navigation enroute to the mission origin or designated rendezvous. Overall mission status becomes "frozen."

2. Uncertainty

Uncertainty plays a significant role in a system such as the Mission Executor. In fact the primary reason for using a forward-chaining rule-based tool such as CLIPS is that there is some knowledge but a great deal of uncertainty about the external environment. What *is* known about the environment can best be classified in heuristics. A specific area of uncertainty that the Mission Executor must reason about is the presence of obstacles. Report of an obstacle at short range automatically generates a command from the executor (emergency situation) but report of an obstacle at the limit of the sonar is a different matter. The obstacle is assigned a confidence factor which comes from the Sonar Processing Suite. Obstacles of high or medium

confidence cause the path to be replanned. The rationale is that the farther away an obstacle is detected, the less radical a turn is necessary. This often results in less deviation from the original track, saving both mission time and energy consumption.

D. MISSION DOCUMENTATION AND OBJECT PERSISTENCE

1. The Need for High Level Mission Documentation

There is a vital need for documentation of AUV missions. All of the AUV projects now in development at various facilities around the country have come to rely on some data recorded onboard the AUV. This compilation of data is valuable for several reasons:

- it can be analyzed by human AUV researchers to update and refine the AUV control systems (both hardware and software)
- it can provide an idea of what works with rule-based systems and where failure in reasoning occurs.
- it can be used as a persistent base of knowledge for "training" AUV's in situation assessment (this was also a conclusion of Westneat (Westneat 1990, pp. 27-33)).

Documentation already exists within the NPS AUV II Baseline system in the form of the Environmental Database which contains some navigational data and data about obstacles which might be encountered. A mission log is maintained by the Navigator module in much the same way that a mission log is kept by the navigator of a maritime vessel. However, in order to adequately study high-level control, a mission log must also be kept of high-level decisions. It can be regarded as a form of captain's log which records the state of the mission at the highest level and justifications for

decisions made. At a standard time interval or whenever the overall mission decision changes, an entry is made to the log. This is accomplished by saving objects and facts to the log file.

2. Object and Fact Persistence in the Executor

Object persistence in a database refers to longevity, its ability to exceed the life of the executing application program. A knowledge base no longer exists at the conclusion of an execution. To save its knowledge, the information must be loaded to a file. Objects are saved via the *save-instances* command. Facts can also be saved by the *save* command (NASA 1991, pp. 169, 188).

VI. PROTOTYPE IMPLEMENTATION AND SIMULATION

This chapter describes the actual prototype implementation. Test results are discussed at the conclusion of the chapter.

A. CONTROL CONSTRUCTS AND OBJECT IMPLEMENTATIONS

The Mission Executor implementation is built around the overall mission state existing in one of three forms: Continue_unrestricted, Continue_with_restrictions, or Abort_Mission. Continue_unrestricted is the initial default state outlined in Chapter five. This state only exists when no functional area is critical or experiencing failure. Most of the rules in the Executor are based on missions which cannot remain in the ideal state due to a casualty or discrepancy in the mission, vehicle, or environmental worlds.

The vehicle reasoning system is implemented upon the download of the mission plan. This triggers the rule Mission_Timer, which continually binds the mission time to the current central processing unit (cpu) time. A timer flag is continually asserted in this rule and retracted in the timer manager rule. The timer manager continually asserts facts which trigger other polling rules. While the detailed implementation of this is available in Appendix A, the main algorithm is shown below:

```
read (desired_scenario);
 if vehicle status = operational, then
    open(mission_file);

   while not end-of-file
     read (mission_file);
```

```
      make posture_object (delimited mission dataline);
    end while;
    initialize vehicle-sensor, mission, environmental, maneuvering,
          navigation statuses;
    initialize mission_timer;
  end if;
  while not terminating condition
    = (completed mission, abort to rendezvous, abort for dynamic recovery)
    mission_time := (current cpu time - mission start time);
    if the mission_time := time of some event then
      instantiate(the event);
    if the mission_time := the appropriate documentation time interval then
      document the mission;
    allow maneuvering, vehicle-sensor, mission, environmental, and navigation
      rules to handle any exceptions to closed loop navigation as they occur;
    propagate changes (functional area supervisors);
    assess impact of any changes;
    propagate recovery or abort configurations;
  end while;
```

Initial development of the Executor actually focussed on the internal world. Coincidentally, it somewhat resembles the model used by Giarratano for his small intelligent database outlined in the CLIPS Objects Manual (Giarratano 1991a, pp. 150-161). Vehicle internal state is modeled in the module **sensor.clp** in which all onboard equipments are represented as objects. The main class which defines an equipment object is SYSTEM_MONITOR. Since there are no actual instances of this object, SYSTEM_MONITOR is an abstract class. From it are derived the various equipments. The structure of the class inheritance hierarchy is discussed in Chapter V. The SYSTEM_MONITOR class takes the form:

```
(defclass  SYSTEM_MONITOR (is-a USER)
   (slot   type_of_reading)
   (slot   reading)
   (slot   status  (default normal))
   (slot   Redundant_Equipment (initialize-only))
   (slot   redline_high    (initialize-only))
   (slot   guardline_high  (initialize-only))
   (slot   guardline_low   (initialize-only))
   (slot   redline_low     (initialize-only)))
```

The abstract class SYSTEM_MONITOR shown above is composed of slots which describe the most general form of equipment sensor onboard. This is easily configurable for various subclasses. The slot type_of_reading is common across all subclasses, as are the reading (the current reading recorded and propagated by the analog-to-digital converter), and the status. The slot Redundant_Equipment is elaborated in the instance declarations. It either establishes an equipment as redundant with a similar or backup equipment, or it takes on the value NONE. Most equipment has a redline reading (either high or low) indicating that the failure point or equipment shutdown limit has been exceeded. The guardline slots exist to provide the equipment to degrade more gracefully, perhaps initiating the turn-key operation to energize the redundant equipment or power source. Naturally, not all equipment or power sources have both high and low limits. The slots which are not applicable can be set to NONE in the subclass definition where the message-handlers which depend on the various slots are elaborated.

The various subclasses of SYSTEM_MONITOR have their own class definitions and respective message-handlers which operate on instances of those classes. Most of the message-handlers in this module are of the daemon variety. These message

handlers are activated when a basic action such as insertion of a new value in an object slot, deletion of a slot value, or reading of a slot value is performed (NASA 1991, pp. 86-87). In this case reading of a slot value is done by a polling rule, monitor-health-continuously. If the value read exceeds a guardline value, then it often places the system being monitored in the condition of *critical.* If the sensor redline value is exceeded, the equipment is assumed to have failed. In the case of a vehicle control system such as the rudder or diving planes, there is also a message-handler which checks the *response* of the system. This often means positional response. If, for example, the autopilot generates a command to turn left and the rudder moves in the wrong direction, then the system is assumed to have become critical. An example of the CONTROL_SYSTEM class and two message-handlers follows:

```
(defclass CONTROL_SYSTEM (is-a SYSTEM_MONITOR)
   (slot     type_of_reading (default potential_in_volts))
   (slot     control-type)
   (slot          response (default normal))
   (message-handler get-reading)
   (message-handler get-response)



(defmessage-handler CONTROL_SYSTEM get-reading after ()
  (bind ?control (instance-name-to-symbol (instance-name ?self)))
  (if (or (and  (> ?self:reading   ?self:guardline_high)
       (< ?self:reading   ?self:redline_high)
            (and (< ?self:reading   ?self:guardline_low)
                 (> ?self:reading   ?self:redline_low))) then
  (assert (Equipment_Critical Control_System ?control))

  else
       (if  (or  (>  ?self:reading  ?self:redline_high)
                 (<  ?self:reading  ?self:redline_low)) then
            (assert (Equipment_Failure Control_System ?control))
       (send ?self put-status INOPERATIVE))))
```

```
(defmessage-handler CONTROL_SYSTEM get-response  after ()
  (if (neq  ?self:response normal) then
  (assert (Equipment_Critical Control_System ?self))))
```

Using low salience, these message-handlers are polled by a rule which uses an object query *do-for-all-instances* for each subclass of SYSTEM_MONITOR interface with rules which determine if a situation is applicable to the failure or critical situation. Low level rules which determine the situation often have the most complex heuristics in the Executor. However, the equipment status rules are uncomplicated, as evidenced by the following:

```
(defrule Control_System_Failure
  (Equipment_Failure Control_System ?control)
  =>
  (if (eq ?control Hover-Thrusters) then
     (assert (Equipment_Mission_Essential no))
  else
   (assert (Equipment_Mission_Essential yes)))
  (assert (Equipment_Status-Assess )))
```

This simply says that any failure of a control system, unless the control system is the hover-thrusters, should be considered mission-essential and that requires impact assessment of the equipment functional area. The assertion of the Equipment_ Mission_Essential fact and Equipment_Status-Assess control fact will trigger an equipment status assessment. If the equipment failure is a failure of the hover thrusters, it will simply be noted.

Objects are not only used to model equipments, but also decisions. Decisions are maintained for purposes of later retrieval in reconstructing the mission and in conducting any possible machine learning for the AUV. The current decision is kept

in an instance called *current*. Whenever a new decision has been made, it is passed to the function decision-change which copies the old decision to an object and in turn replaces all the characteristic slots of the current decision. Maintaining the decision is useful not only for mission documentation, but also in resolving conflicts between states. The decision objects and function constructs take the following form:

```
(deffunction decision-change  (?the_type ?the_rule ?the_level ?the_action)
  (bind ?name (gensym*))
  (make-instance ?name of DECISION)
  (copy-old-instance ?name of DECISION)
  (send [current] put-type ?the_type)
  (send [current] put-rule ?the_rule)
   (send [current] put_action ?the_action)
  (send [current] put-decision_time ?*mission_time*))


(deffunction copy-old-instance (?instance)
  (send (symbol-to-instance-name ?instance) put-type
    (send [current] get-type ))
  (send (symbol-to-instance-name ?instance) put-level
    (send [current] get-level))
  (send (symbol-to-instance-name ?instance) put-action
    (send [current] get-action))
  (send  (symbol-to-instance-name ?instance) put-decision-time
    (send [current] get-decision_time)))


(defclass DECISION (is-a USER)
  (slot type)
  (slot rule)
  (slot level)
  (slot action)
  (slot decision_time))
```

Mission documentation is is actually maintained by a rule which uses the *save-instances* and *save-facts* commands to save the mission-state at that point. This is done at a specified time interval, usually every twenty seconds. Facts and instances normally

cannot be saved together in the same file using save-facts and save-instances, so that there is both an instances log and a facts log.

```
(defrule Document_Mission
 ?document <-  (document mission)
=>
(if (> ?*mission_time* ?*Time_Interval*) then
     (save-instances "Mission_Log.ins")
(save-facts "Mission_Log.facts")
(bind ?*Time_Interval* (+ ?*Time_Interval* 20.0)))
(retract ?document))
```

Simulation events are also modeled as objects. An event is made up of its number, its time of instantiation, the event trigger (a fact assertion or instance message sent to a handler), and a description of the event for output. The rule trigger is actually a literal string kept in the event_action slot of the object EVENT_SCHEDULE. When the event is activated, the CLIPS *eval* function is used to instantiate the fact or object message. The global variable ?*current_event* updates the focus to the next current event. The event is actually instantiated by activating all the events whose event times have passed and have not yet been activated. The output lines shown in Appendix A have been omitted here for clarity in understanding the rule/message-handler interaction:

```
(defclass EVENT_SCHEDULE (is-a USER)
   (slot   event_no)
   (slot   event_time)
   (slot   event_action)
   (slot   description)
   (message-handler execute event)
```

```
(defmessage-handler EVENT_SCHEDULE execute-event primary ()
  (eval ?self:event_action)
  (bind ?*current_event* (+ ?*current_event*  1)))
```

```
(defrule event_schedule_manager
  (declare (salience -500)
  ?event <- (schedule_event next_event)
=>
  (do-for-instance  ((?event EVENT_SCHEDULE))
        (and (< ?event:event_time ?*mission_time*)
             (eq ?event:event_no ?*current_event*))
  (send ?event execute-event))
  (retract ?event))
```

B. LAYERING OF RULES

As described in the previous chapter, rules in SKIPPER are layered according to level of reasoning. The lowest-level rules actually carry out the corrective action by ordering Guidance to turn left or ascend-to-safe-depth or ascend-24 (signifying ascend ten inches). This is a significant break from a human paradigm. In a naval vessel where maneuvering control is conducted by humans, no human controller is assumed to be faultlessly competent. The commanding officer frequently cross-checks verbal reports and orders to ensure that his instructions have been carried out. This is of particular consequence in a special maneuvering situation. In SKIPPER, the lower level rules are assumed to be competent operators or controllers. For example, the maneuvering rule abnormal_dive is given the responsibility to order Guidance to decrease the speed and ascend to the designated safe depth, bringing the vessel to a safe configuration before it propagates this situation to the intermediate level assessment rule:

```
(defrule abnormal_dive
  (configuration ?config)
  (action   dive)
  (or  (Equipment_Failure Control_System Plane_Controls)
       (obstacle_clearnace ?clearance&:(neq ?clearance normal))

=>
```

```
(Call-Guidance-Command  Decrease-Speed Drive-Motors)
(Call-Guidance-Command  Ascend-?*safe_depth* Planes)
(assert (maneuvering_ability Major_Restriction)
(assert (Maneuvering_Status_Assess)))
```

The intermediate level rules appear to be candidates for conflict with lower level rules. Because the overall mission status is dependent on rapid propagation of changes from the assessment rules, the assessment rules are given a higher salience value. Some experiments with the artificial neural system paradigm demonstrated that dynamic salience is not always effective. In short, the focus of consistently increasing salience in a particular area based on past inputs can lead to a delay in other functional areas. This can be critical if the other functional area is about to fail although it had no previous record of doing so. Assigning a higher salience value to the assessment rules gives them adequate priority. The Maneuvering Status Assessment rule which handles an equipment failure and is linked to the low level rule above is an illustration of this:

```
(defrule Maneuvering_Status_Assessment_EquipmentFailure
  (declare  (salience ?*maneuver_salience*))
  (Equipment_Failure  Control_System ?control&:(neq ?control Hover-Thrusters))

=>

  (assert  (decision-change maneuvering_assessment
          Maneuvering_Status_Assessment_EquipmentFailure  assessment
         propagate_equipment_failure))

  (assert (Maneuvering_Status severely_restricted)))
```

Its sibling rule, which evaluates other types of maneuvering status problems, tabulates the number of discrepancies. A certain number of discrepancies signals that

the navigational track is too ambitious, requiring an unacceptable number of obstacle avoidance maneuvers. The discrepancies are bound to the global variable ?*maneuverability_factor* which triggers the Maneuvering_Status_Assessment rule and eventually causes a mission abort.

The overall mission assessor examines the current status of all functional areas and makes a determination on the state of the vehicle mission. At that point, the overall mission status is changed, if necessary, and the results propagated down to the respective mission abort or mission restricted rules. Because of the length of this rule and its respective function, they are displayed in **Figure 6-1**.

All of the functional areas have a similar structure. Maneuvering has the added feature of low level assessment rules which examine the obstacle object base to see if the indicated obstacles pose a collision danger. This added assessment requires examination of several object slots and some tabulations, actions which lead to increased overhead. This overhead is clearly observed in the simulation runs described below.

C. USE OF FUZZY LOGIC AND TRUTH MAINTENANCE

Truth maintenance is an integral part of the mission executor, mostly in the highest levels. The logical construct described previously in chapter three is the CLIPS environment-installed method of maintaining the integrity of the state. The vehicle's initial state (hence ideal state) rests upon a foundation of all functional areas being operational. This does not mean that all functional areas are devoid of any

```
(defrule Overall_Mission_Assessor
  ?overall <- (Overall_mission_status ?status)
  ?equip <- (Equipment_Status ?equipment_status )
  (Maneuvering_Status ?maneuver_status )
  (Navigation_Status  ?nav_status )
  (Environmental_Status ?environment_status )
  (Spec_Mission_Status ?specmission_status )
   ?change <- (propagate change)
 =>
 (retract ?change)
(if (eq ?equipment_status major_failure) then
    (bind ?*Functional_area_failure* (+ ?*Functional_area_failure* 1))
else
   (if (eq ?equipment_status equipment_critical) then
      (bind ?*Functional_area_critical* (+ ?*Functional_area_critical*
                                1))))
  (if (eq ?maneuver_status severely_restricted) then
     (bind ?*Functional_area_failure* (+ ?*Functional_area_failure*
                                1))
   else
      (if (eq ?maneuver_status restricted) then
         (bind ?*Functional_area_critical*
               (+ ?*Functional_area_critical* 1))))
     (if (eq ?nav_status out_of_tolerance) then
        (bind ?*Functional_area_failure* (+ ?*Functional_area_failure*
                                1))
       else
        (if (eq ?nav_status critical) then
           (bind ?*Functional_area_critical*
                 (+ ?*Functional_area_critical* 1))))
        (if (eq ?environment_status major_deviation) then
           (bind ?*Functional_area_failure*
                 (+ ?*Functional_area_failure* 1))
        else
       (if (eq ?environment_status critical_deviation) then
          (bind ?*Functional_area_critical*
                (+ ?*Functional_area_critical* 1))))
     (if (eq ?specmission_status  infeasible) then
        (bind ?*Functional_area_failure*
         (+ ?*Functional_area_failure* 1)))
 (Total-Functional-Problems ?overall))
```

Figure 6-1. Overall Mission Assessment Rule

complications, just that the complications will not cause the vehicle to become critical. The essence of this ideal state is embodied in the following rule:

```
(defrule Continue-Mission_unrestricted
  (logical   (Equipment_Status  normal)
             (Maneuvering_Status  unrestricted)
             (Environment_Status  normal)
             (Navigation_Status  within_tolerance)
             (Spec_Mission_Status  feasible))
 =>
  (decision-change Overall_Mission Continue-Mission_unrestricted
       High  Continue-Mission-with-no-restrictsions)
  (assert (Overall_mission_status Continue_Unrestricted)))
```

Any failure of a particular functional area will cause the mission status to be retracted. However, the functional area which caused the change in overall mission status will cause the overall mission status to change. Thus, just as the overall mission status of Continue_unrestricted is being retracted, a new mission state is being asserted. There is no "stateless" gap in mission status.

A functional area failure causes a mission abort, resulting in vehicle recovery or an abort transit to the designated rendezvous. The abort status is one that should remain in effect until the vehicle is recovered. However, in the interval between the status change and the actual vehicle recovery, there is a possibility that a functional area becoming critical could later attempt to cause a status of Continue_with_Restrictions. There is also the possiblity that the functional area recovery rules could cause a new state of Continue_unrestricted. To counter any possibility that this could happen, a truth maintenance feature of status lock is incorporated. This causes the mission assessor rule to be excised or removed. Thus, no mission state change can occur. This rule is depicted below:

```
(defrule Abort_Mission
 (declare (salience 500))
 (Overall_mission_status  Abort_mission)
 ?change <- (propagate-change down)
 ?point  <- (waypoint ?no)
=>
 (retract ?point)
 (retract ?change)
 (decision-change Overall_Mission Abort_Mission Low
  lock_status_and_replan_route_to_abort_rendezvous)
 (undefrule Overall_Mission_Assessor)
 (do-for-instance ((?control CONTROL_SYSTEM))
                  (and  (eq ?control:status INOPERATIVE)
                        (neq ?control Hover-Thrusters))
 (progn  (Call-Guidance-Command turn_on_transponder transponder
         (Call-Guidance-Command ascend_surface planes)
 (printout t crlf crlf ">>>>> Shutting Down for Dynamic Recovery <<<<<" crlf
   ">>>>> Transponder will function for 2 hours <<<<<" crlf)
  (halt)))
 (Abort-Route))
```

Fuzzy logic is used in obstacle avoidance rules in the confidence factor assignment. If the confidence factor is high to medium and the obstacle is within the 180 degree arc about the bow of the AUV, then the obstacle is considered to be a collision danger. This confidence factor is checked whenever an obstacle alert flag is sent, be it an update or a new obstacle.

```
(defmessage-handler OBSTACLE  obstacle-change primary ()
 (if (and (eq ?self:confidence_factor high)
          (eq ?self:confidence_factor medium))
          (or  (and (>= ?self:bearing 270.0) (<= ?self:bearing 359.))
               (and (<= ?self:bearing 90.0)  (>= ?self:bearing 0.)))) then
   (send ?self put-collision_danger YES)
   (assert (collective_obstacle_assessment))
 else
   (send ?self put-collision_danger NO)))
```

D. RESULTS

Implementing the Mission Executor Code involved some testing of the rules to determine if the overall desired terminal action could be generated. In this heuristic

model, the intent was to determine if symbolic high-level reasoning would achieve the desired behavior. Another benefit was to determine if the reasoning system could recognize situations and try to approximate real-time constrained decision-making. Navigational waypoints used in the scenarios are based on the model of the Naval Postgraduate School pool by Magrino and Floyd show in **Figure 6-2**. These are the same used in evaluating the navigational controller (Magrino, 1991). An average mission time of two to four minutes is used for the ideal non-avoidance path transit/mission. The scenarios described are listed in Appendix B for reference. Propagation effects are displayed in **Table 6-1**. Run-times do not agree with mission-completion times simply because mission times are based on a starting time which is instantiated upon the full download of the mission navigational plan, often a full 2.0 seconds or more after the beginning of program execution.

Scenario one merely tested the most basic case, pre-planned mission execution monitoring (waypoint sequencing). The Autonomous Underwater Vehicle (AUV) was given a set of waypoints, each with its specified estimated time of arrival as a constraint. At the third waypoint, the AUV missed its time constraint by a considerable amount (47 seconds), enough to cause the Waypoint_DistanceTime_Check rule to alert the navigation assessment rule. A time difference of 20.0 to 39.99 seconds is considered to be minor, resulting only in a command to Guidance to increase the speed. A time difference of 40.0 seconds or more is considered to be a major time deviation, resulting in a command to increase speed. The Navigation Assessment rule uses the heuristic rule that four navigation problems such as this cause a replan of the

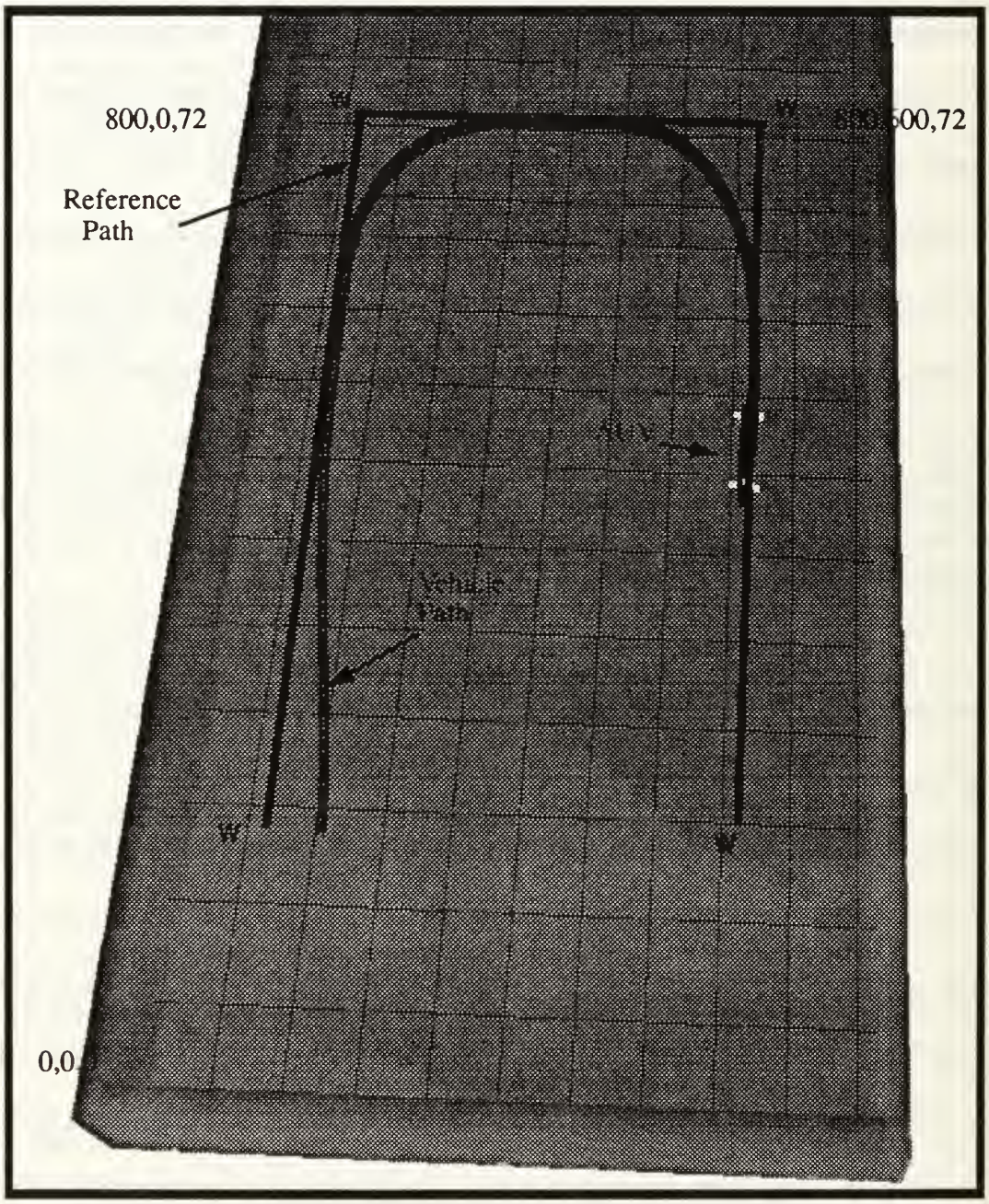


Figure 6-2. NPS Pool Mission Schematic

TABLE 6-1 SCENARIO RESULTS					
Major Status Change	Scen 1 reaction/ propagation times(secs)t	Scen 2	Scen 3	Scen 4	Scen 5
Recognition	0.276	0.452	0.243	0.244	----
Assessment	0.35	6.55	0.17	0.16	0.303
Overall Change	0.16	0.14	0.16	0.15	0.17

navigational waypoint plan. With only one navigation problem, this resulted in no true change in the navigation status. Nonetheless, the navigation status was assessed for any possible effect. The only effect was the low-level command to increase speed although the current navigation status was propagated to the overall mission assessor. Recognition of a large navigational discrepancy in time resulted in a time of propagation of 0.28 seconds from the Waypoint-DistanceTime_Check rule to the Navigation Assessment Rule. Recognition that this was not a change to status took 0.16 seconds. The overall elapsed mission time was 3 minutes 30 seconds with 18365 rules being fired.

Scenario two tested the ability of SKIPPER to recognize an untenable obstacle avoidance situation. Both short range obstacles and long-range obstacles were tested. The first recognition of an obstacle close-aboard led to an ascent to safe-depth. This also tested a rule recognizing possible shoaling or grounding of the vessel. The emergency avoidance maneuver rule began its time check of the avoidance maneuver. An obstacle detected at long range led to assessment of the obstacle as threatening to the AUV. The overall maneuvering status was changed to Continue_with_Restrictions. At one point enough obstacles had accumulated to cause the collective obstacle assessment rule to characterize the situation as involving a critical number of obstacles (heuristic used is four separate encounters). Later the collective obstacle assessment rule determined that the critical point had been breached by accumulation of too many obstacles along the track (the heuristic here is that too many obstacles will cause too many time-consuming avoidance maneuvers). The maneuvering status assessment rule

determined that this was a functional area failure. The maneuvering functional area failure then forced recognition that this was an abort-mission situation. From recognition of the critical point at 50.45 seconds into the mission, it took approximately 6.55 seconds to recognize that this was an undesirable situation. The change in the maneuvering status and subsequent overall assessment of the mission resulted in a time of propagation of 0.14 seconds.

Scenario three involved a vehicle control system failure. After passing several waypoints, the AUV experienced an electrical failure of the diving planes. The first result was a failure of maneuvering status because that was the more specific rule. The control system failure rule fired shortly after that leading to an overall mission assessment that this was an abort situation. From the instantiation of the triggering event until the time it was recognized as an abort situation was an interval of 0.24 seconds. Propagation of the maneuvering status or equipment status to the overall mission assessor is difficult to absolutely determine because of the fact that both maneuvering assessment and equipment status assessment fired. Either one could have caused the overall mission status to change. Because of the high salience of both rules, activation of the overall mission assessor occurred only 0.17 seconds after the equipment status assessment rule fired.

Scenario four evaluated both some obstacle avoidance and environmental phenomena. Only two obstacle encounters were realized, resulting in only minor deviations to the planned navigational track. A significant environmental phenomena was simulated by having readings in all three environmental sensors exceed allowable

limits. This resulted in a mission abort. From the time of the triggering event until the recognition by the mission assessor that it was an abort situation, 0.56 seconds elapsed.

Scenario five tested multiple equipment failures. The AUV passed through several waypoints missing only one time constraint. A sonar failure (forward sonar) led to a reduction in the overall mission status to *Continue_with_Restrictions* as the sonar went to a critical state. A second sonar (port sonar) led to a reinforcement of that state. Failure of the rudder finally led to the AUV surfacing and energizing its transponder. From the triggering event until the decision to abort, 0.47 seconds elapsed.

E. EVALUATION

Comparison of results reveals that propagation of status from the functional area assessors to the overall mission status assessor will probably meet real-time constraints in the relatively slow-moving environment of the AUV in its testing facility. The true time dependency does appear to be in the low-level action or assessment rules. Situation recognition depends on good heuristics. Using an artificial neural paradigm in which assessment rules were placed on the agenda more quickly based on previous assessment rule firings (and dynamic salience) did not appreciably increase the speed with which propagation of the state occurred. In fact, in at least one situation the propagation speed was slowed by 0.5 seconds.

The use of a layered situation-based reasoning system appears to be sound. By using an intermediate level assessment rule, the desired rapid reaction can be taken at the low-level and the assessment of functional state can proceed at the same time.

Thus, there need not be a salience assigned to every level. This tends to diminish the benefit of a rule-based system. While it does not appear to work well in this implementation, a dynamic salience may be beneficial to focus on desired reactions when the Mission Executor is interfaced with an updated version of the Guidance system which can handle interrupt commands. Refinement of heuristics will certainly be necessary to further optimize the rule base.

VII. CONCLUSION AND RECOMMENDATIONS

A. SUMMARY OF RESULTS AND CONTRIBUTIONS

1. A Prototype Expert System for Mission Execution

A small prototype has been designed, implemented, and tested for several scenarios. While not all possible scenarios could be tested, experience in testing and debugging the Mission Executor implemented in CLIPS version 5.0 illustrates the rapid prototyping capabilities that are available and the great utility of objects to represent the onboard systems. Rules for newly-envisioned situations can be added with relative ease. Thus, the prototype is easily extensible.

2. Software Architecture for Mission Execution

The hierarchical structure designed has a recognizable data flow. The incorporation of the *status-lock* feature by using the *undefrule* command to freeze a mission state and prevent state/rule collision is an effective tool for extension in other areas. Status lock can be an effective tool for debugging other types of programs in which a final overall state must be maintained while other final lower level actions are executing.

3. Determination of Guidance Interrupt Commands

An initial attempt at defining Guidance interrupt commands has been accomplished and will subsequently be refined with more experience in submarine maneuvering.

4. Identification of New Data Flow in the Baseline System

The ability of the Executor to get further navigation updates after a collision_avoidance maneuver which takes it from the desired path indicates a new possible on-demand data flow from the Navigator to the Executor. Further, it appears reasonable that Guidance should provide some kind of confirmation that it has carried out an interrupt command.

B. FUTURE WORK

Research into a configurable mission executor has several areas for extension. This mission executor implementation is relatively immature, and further experience in small underwater vessel missions will allow for greater refinement of its rule base.

1. Mission Executor Portability

The Mission Executor has several modes in which it can reside onboard the GESPAC computer. As mentioned in Chapter III, a CLIPS executable module can be created by changing various flags in the CLIPS C language source code and recompiling the Executor application. Another possible alternative is to embed the Executor application in a large shell program which would hold all of the modules. While the two previous suggestions would result in a storage savings, the best solution

for the Executor is to port the entire CLIPS interpreter (with the exception of development tools) to the onboard computer. This will allow for greater flexibility in the form of use of the *build* and *eval* functions to construct rules as the mission is in progress.

The build and eval functions can be very useful in coercing the AUV to "learn" about difficulties encountered along the designated track. A collective decisions rule can be invoked to analyze all of the decisions made thus far (previously archived in the decision objects).

2. Interfacing the Executor to Dependent Modules

Although interfaces to the various dependent modules are discussed to some extent in Chapter IV, some of the interfaces will remain hypothetical until all of the dependent modules are completed. Naturally, incorporation of the executor into the overall system will require that a comprehensive system alteration plan be developed. The CLIPS-to-Ada and constructs-to-c external interfaces need to be defined.

3. Porting the Executor to the AUV II Graphical Simulator

As the offboard mission planner is completed, the actual porting of CLIPS source code to the updated AUV II simulator will take place. While this in itself should not be tremendously difficult, methods of simulating casualties visually on the IRIS machine need to be developed so that SKIPPER can give a more intuitive representation of its abilities.

4. Incorporation of Specialized Mission Rules

At present, the AUV operates in a constrained testing environment, the Naval Postgraduate School swimming pool. Research for some time to come will focus primarily on transit, avoidance of obstacles and other hazards, vision and sonar sensing, and safe return of the vehicle. Eventually, the vehicle will be able to carry out a very basic mission such as deploying a camera or a hydrographic instrument for a specified period of time. Many possible AUV missions are elaborated in (MacPherson, 1988). Rules need to be incorporated for the situations described in that research which cover casualties, environmental degradations, and obstacles, all of which could hinder or hazard the specialized mission.

LIST OF REFERENCES

(Arthur 1991)
Arthur, T. and Pokrant, M., "Desert Storm at Sea," U. S. Naval Institute *Proceedings*, Vol. 118/5/1, May 1991.

(Bellingham 1990)
Bellingham, J. G. and Consi, T. R., "Robots Underwater - Ongoing Research at MIT Sea Grant," *Sea Technology*, Vol. 31, No. 5, May 1990.

(Bellingham 1990a)
Bellingham, J. G. and Consi, T. R., "State Configured Layered Control," *Proceedings of the IARP 1st Workshop on Mobile Robots for Subsea Environments*, Monterey, California, October 23-26, 1990.

(Blidberg 1990)
Blidberg, D. R. et al., "The EAVE AUV Program at The Marine Systems Engineering Laboratory," *Proceedings of the IARP 1st Workshop on Mobile Robots for Subsea Environments*, Monterey, California, October 23-26, 1990.

(Bonsignore 1991)
Bonsignore, J., *Underwater Multidimensional Path Planning for the NPS AUV II Autonomous Underwater Vehicle*, Masters's Thesis, Naval Postgraduate School, Monterey, California, September 1991.

(Bowen 1990)
Bowen, P. S., Chappell, S. G., and Gonzalez, R., "Using Common LISP in the EAVE Autonomous Underwater Vehicle," *Journal of Oceanic Engineering*, Vol. 15, No. 3, July 1990.

(Brooks 1986)
Brooks, R. A., "A Layered Intelligent Control System for a Mobile Robot," *Robotics Research*, Fanguera and Girald, eds., Cambridge: MIT Press, 1986, pp. 365-372.

(Busby 1990)
Busby, F. and Vadus, J. R., "Autonomous Underwater Vehicle R&D Trends," *Sea Technology*, Vol 31., No. 5, May 1990.

(Caddell 1991)
Caddell, T., *Three-Dimensional Path Planning for an Undersea Environment*, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1991.

(Cloutier 1990)
Cloutier, M. J., *Guidance and Control System for an Autonomous Vehicle*, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1990.

(Floyd 1991)
Floyd, C. A., *Design and Implementation of a Collision Avoidance System for the NPS Autonomous Underwater Vehicle (AUV II) Utilizing Ultrasonic Sensors*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1991.

(Giarratano 1991)
Giarratano, *CLIPS User's Guide: Volume 1 Rules*, Software Technology Branch, Lyndon B. Johnson Space Center, Houston, Texas, January 1991.

(Giarratano 1991a)
Giarratano, *CLIPS User's Guide: Volume 2 Objects*, Software Technology Branch, Lyndon B. Johnson Space Center, Houston, Texas, May 1991.

(Healy 1990)
Healy, A. J. et al., "Planning, Navigation, Dynamics and Control of Autonomous Underwater Vehicles," Proposal for Research submitted to Naval Surface Weapons Center, White Oak Laboratories, Naval Postgraduate School, May 1990.

(Healy 1990a)
Healy, A. J. et al., "Mission Planning, Execution and Data Analysis for the NPS AUV II Autonomous Underwater Vehicle," *Proceedings of the IARP 1st Workshop on Mobile Robots for Subsea Environments*, Monterey, California, October 1990.

(Isik 1990)
Isik, C. and Meystel, A. M., "Pilot Level of A Hierarchical Controller for an Unmanned Mobile Robot," *IEEE Journal of Robotics and Automation*, Vol. 4, No 3, Winter 1989.

(McPherson 1988)
MacPherson, D. L., *A Computer Simulation Study of Rule-based Control of an Autonomous Underwater Vehicle*, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1988.

(Meijer 1991)
Meijer, G.R. et al., "Exception Handling System for Autonomous Robots based on PES," *Proceedings of the International Conference on Intelligent Autonomous Systems,* Amsterdam, the Netherlands, December 1989.

(Mettrey 1991)
Mettrey, "A Comparative Evaluation of Expert Systems Tools," *IEEE Computer*, February 1991.

(NASA 1991)
CLIPS Reference Manual Volume I, Basic Programming Guide, Software Technology Branch, Lyndon B. Johnson Space Center, Houston, Texas, 1991.

(NASA 1991a)
CLIPS Reference Manual Volume II, Advanced Programming Guide, Software Technology Branch, Lyndon B. Johnson Space Center, Houston, Texas, 1991.

(Ong 1990)
Ong, S. M., *A Mission-Planning Expert System with Three-Dimensional Path Optimization for the NPS Model 2 Autonomous Underwater Vehicle*, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1990.

(Polmar 1991)
Polmar, N., "Robot Submarines," *U. S. Naval Institute Proceedings*, Vol. 118/9/1, September, 1991.

(Riley, 1987)
Riley, G. et al., "CLIPS: An Expert System Tool for Delivery and Training," *Proceedings of the Third Conference on Artificial Intelligence for Space Applications*, NASA Conference Publication 2492, Huntsville, Alabama, November 2-3, 1987.

(Rodseth 1990)
Rodseth, O. J., "Object-Oriented Software for AUV Control," *Proceedings of the IARP 1st Workshop on Mobile Robots for Subsea Environments*, Monterey, California, October 23-26, 1990.

(Schudy 1990)
Schudy, R. B. and Duarte, C. N., "Advanced Autonomous Underwater Vehicle Software Architecture," *Proceedings of the Symposium on Autonomous Underwater Vehicle Technology*, Washington, DC, June 5-6, 1990.

(Spelt, 1989)
Spelt, P. F. et al., "Learning by an Autonomous Robot at a Process Control Panel," *IEEE Expert*, Winter 1989.

(Vezina 1988)
Vezina, J. M. and Sterling, L., "A CLIPS Prototype for Autonomous Power System Control," *Proceedings of the Fourth Conference on Artificial Intelligence for Space Applications*, NASA Conference Publication 3013, Huntsville, Alabama, November 15-16, 1988.

(Westneat 1990)
Westneat, A. S. and Clearwaters, S. L., "A Generalized Alternative Contingency Matrix for the Autonomous Untethered Vehicle (AUV)," *Proceedings of the Symposium on Autonomous Underwater Vehicle Technology*, Washington, DC, June 5-6, 1990.

(Zheng 1990)
Zheng, X., Jackson, E., and Kao, M., "Object-Oriented Software Architecture for Mission-Configurable Robots," *Proceedings of the IARP 1st Workshop on Mobile Robots for Subsea Environments*, Monterey, California, October 23-26, 1990.

APPENDIX A. MISSION EXECUTOR SOURCE CODE

```
;    Programmer        :    W P Wilkinson
;    System            :    CLIPS 5.0
;    Program           :    AUV Mission Executor "Skipper"
;    Functional Area   :    Main Program
;    Latest Revision   :    21 August 91
; ---------------------------------------------------------------; :
;                          Description
   :  The AUV Mission Executor System. This module skipper.clp
;  :  is the main program to which all of the other five modules
;  :  are subordinate. The highest reasoning level (overall mission
;  :  assessment) as well as utility rules for saving decisions
;   : reside here. Event management is also controlled here. A continuous
;  :  loop checks for termination events which shutdown the Mission
;  :  Executor.
;  :  This software incorporates the use of the following in the
;  :  "layered worlds" paradigm:
;  :  -- Use of Fuzzy Logic
;  :  -- Prioritization of important actions and state assessment
;  :  -- Truth Maintenance via CLIPS logical constuct and "status lock"
;
:_____________________________________________________________________


;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;   Global variables which pertain to main module or to all    ;;;
;;;   parts of the program.  For the most part the actual values ;;
;;;   are unimportant to understanding of the program.           ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

   (defglobal ?*start_time*                =      0.
              ?*mission_time*              =      0
              ?*mission_degradation_time*  =      0.
              ?*recovery_time*             =     30.0
              ?*Time_Interval*             =     20.0
              ?*emergency_salience*        =   1000
              ?*mission_critical_power*    =     30.0
              ?*Functional_area_failure*   =     0
              ?*Functional_area_critical*  =     0
              ?*current_event*             =     1
              ?*Goalx*                     =     0.0
              ?*Goaly*                     =     0.0
              ?*Goalz*                     =     0.0 )
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                Function  show-demo-description                  ;;;
;;;  Function which shows the user a selection of scenarios.        ;;;
;;;  It is by no means all-encompassing.The 55 rules which make;;;
;;;  up this system can be permuted to build many scenarios         ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

  (deffunction show-demo-description ()
     (printout t crlf crlf crlf crlf crlf crlf)
     (printout t "                Welcome to the MISSION EXECUTOR DEMO ")
     (printout t crlf crlf  )
    (printout t "WAYPOINTS:   All scenarios take place over the same set"
                                                                  crlf
               "            of INITIAL waypoint coordinates." crlf crlf)
     (printout t "EQUIPMENT:   All equipment is simulated in the event
                                                          file" crlf
                "             Objects are created for each onboard
                                                 equipment" crlf crlf)
     (printout t "SITUATIONS:  All situations are also simulated in
                                                    the event" crlf
                "             file. For instance, an obstacle detection
                                                      is  " crlf
                "             listed and this simulates the Obstacle
                                                   Avoidance" crlf
                "             DecisionMaker passing this information
                                                  through" crlf
                "            the interface to the Executor . " crlf crlf)

     (printout t "SCENARIO CHOICES: select number <Ret>" crlf
                 "1   Waypoint_Hopping Only (transit)"  crlf
                 "2   Obstacle Avoidance    "  crlf
                 "3   Vehicle Control System Failure" crlf
                "4   Obstacles and Environment Problems "           crlf
                 "5   Equipment Failures    "             crlf
                "6   Exit the Simulator   "            crlf crlf crlf))



;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;   Decision  objects and functions of possible use in a machine   ;;;
;;; learning program. The decision can be archived in an object.     ;;
;;; Most importantly, the decisions made in the system are output   ;;;
;;  so that a future developer can see the propagation of changes   ;;;
;;; in decisions.                                                   ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;   This copies the current data in the current decision to a ;;;
;;;;;   a storage object                                           ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

  (deffunction copy-old-instance (?instance)
     (send (symbol-to-instance-name ?instance) put-type
               (send [current] get-type ))
     (send (symbol-to-instance-name ?instance) put-level
               (send [current] get-level))
     (send (symbol-to-instance-name ?instance) put-action
               (send [current] get-action))
     (send (symbol-to-instance-name ?instance) put-decision_time
               (send [current] get-decision_time)))


  (defclass  DECISION (is-a USER)
     (slot type )
     (slot rule)
     (slot level)
     (slot action)
     (slot decision_time))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;  This routine  creates the decision objects and also puts out ;;;
;;;  the propagation trail                                       ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;



  (deffunction decision-change (?the_type ?the_rule ?the_level
?the_action)
     (bind ?name (gensym*))
     (bind ?the_time (- (time) ?*start_time*))
     (make-instance ?name of DECISION)
     (copy-old-instance ?name)
     (send [current] put-type ?the_type)
     (send [current] put-rule ?the_rule)
     (send [current] put-level ?the_level)
     (send [current] put-action ?the_action)
     (send [current] put-decision_time ?the_time)
     (printout t  crlf    ">>>>>>>>>>>> Decision <<<<<<<<<<<<   "  crlf
                        " type   :  " ?the_type   crlf
                        " rule   :  " ?the_rule   crlf
                        " level  :  " ?the_level  crlf
                        " action :  " ?the_action crlf)
     (format t           " time   :   %6.3f%n%n"  ?the_time   ))
```

```
;;;;;;;;;;;;;;;;;  Event Objects and Handler ;;;;;;;;;;;;;;;;;;;;;;;
;;;  Events are modeled as objects with a number, description,   ;;;
;;;  and time. The event description and time are output as they ;;;
;;;  are processed for execution.                                ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defclass  EVENT_SCHEDULE (is-a USER)
    (slot   event_no)
    (slot   event_time)
    (slot   event_action)
    (slot   description)
    (message-handler execute-event))



(defmessage-handler EVENT_SCHEDULE execute-event primary ()
   (eval ?self:event_action)
   (printout t  crlf  "************************************************"
                                                          crlf
                        " Event Number :  "   ?self:event_no      crlf
                 "_____________________________________________" crlf
                " Description :  "  ?self:description             crlf
                "_____________________________________________"
                                                          crlf)
  (format t       " Time     :   %6.3f%n" ?self:event_time  )
  (printout t         "************************************************"
                                                         crlf)
    (bind ?*current_event* (+ ?*current_event*  1)))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;         Navigation  waypoint posture objects             ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


  (defclass POSTURE  (is-a USER)
   (slot configuration)
   (slot action)
   (slot   number)
   (slot   x_pos)
   (slot   y_pos)
   (slot   z_pos)
   (slot  theta )
   (slot  ETA))
```

```
;;;
```

```
;;;
```

```
     (open  "obstacles.dat" obstacles "r")
     (assert (current_plan "mission_plan.dat"))
     (assert (upload plan))
     (set-salience-evaluation every-cycle))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                  Starts the vehicle reasoning system           ;;;
;;;   Loads up the mission reference postures into objects.        ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    (defrule upload
       (vehicle operational)
       ?current <-  (current_plan ?file)
       (waypoint  ?no)
       ?upload <-(upload plan)
       =>

        (if (= ?*start_time* 0.) then
           (bind ?*start_time* (time)))
           (open  ?file  plan "r")
           (bind ?number ?no)
           (bind ?config (read plan))
           (while (neq ?config EOF)
             (bind ?name (gensym*))
              (make-instance ?name of POSTURE
                (configuration  ?config)
                (action         unknown)
                (number ?number)
                (x_pos (read plan))
                (y_pos (read  plan))
                (z_pos (read plan))
                (theta (read plan))
                (ETA   (read plan)))
                (bind ?config (read plan))
                (bind ?number (+ 1 ?number)))
        (close plan)
        (retract ?current)
        (retract ?upload)
        (assert (waypoint-status mark_on_top))
        (assert (mission_timer running))
        (assert (Current_action underway)) )
```

```
;;;
```

```
;;;
```

```
                    (assert (propagate-change down))
                   else
                    ( if (and (eq ?*Functional_area_failure* 0)
                              (neq ?*Functional_area_critical* 0)) then
            (retract ?overall)
           (assert (Overall_mission_status Continue_with_Restrictions))
           (decision-change Overall_Mission Overall_Mission_Assessor
                         High  ContinueMission_with_restrictions)))))


   ;;;;;;;;;;;;;;;;;;;;;;;; Overall Mission Status ;;;;;;;;;;;;;;;;;;
   ;;; This waits on changes to the 5 rule areas  Changes from    ;;;;
   ;;; these are indicated with the assertion of the propagate_   ;;;
   ;;; change flag. Changes to functional areas are checked for   ;;;
   ;;; effect to the overall mission by the function Total-       ;;;
   ;;; functional Problems.                                       ;;;
   ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

   (defrule Overall_Mission_Assessor
      ?overall <- (Overall_mission_status ?status)
      ?equip <- (Equipment_Status ?equipment_status )
      (Maneuvering_Status ?maneuver_status )
      (Navigation_Status  ?nav_status )
      (Environmental_Status ?environment_status )
      (Spec_Mission_Status ?specmission_status )
      ?change <- (propagate change)
    =>
       (retract ?change)
       (if (eq ?equipment_status major_failure) then
          (bind ?*Functional_area_failure* (+ ?*Functional_area_failure*
1))
          else
            (if (eq ?equipment_status equipment_critical) then
               (bind ?*Functional_area_critical*
                        (+ ?*Functional_area_critical* 1))))

            (if (eq ?maneuver_status severely_restricted) then
                  (bind ?*Functional_area_failure*
                                 (+ ?*Functional_area_failure* 1))

              else
               (if (eq ?maneuver_status restricted) then
                     (bind ?*Functional_area_critical*
                        (+ ?*Functional_area_critical* 1))))
               (if (eq ?nav_status out_of_tolerance) then
                   (bind ?*Functional_area_failure*
                               (+ ?*Functional_area_failure* 1))

                    else
                    (if (eq ?nav_status critical) then
                      (bind ?*Functional_area_critical*
```

```
                        (+ ?*Functional_area_critical* 1))))
              (if (eq ?environment_status major_deviation) then
                  (bind ?*Functional_area_failure*
                        (+ ?*Functional_area_failure* 1))
                 else
                  (if (eq ?environment_status critical_deviation) then
                      (bind ?*Functional_area_critical*
                          (+ ?*Functional_area_critical* 1))))
              (if (eq ?specmission_status  infeasible) then
                  (bind ?*Functional_area_failure*
                        (+ ?*Functional_area_failure* 1)))
               (Total-Functional-Problems ?overall))


;;;;;;;;;;;;;;;;;;;;  Unrestricted Mission ;;;;;;;;;;;;;;;;;;;;;;;
;;;  Default Status for start of mission and when the status ;;;
;;;  is restored to normal after a recovery from mission     ;;;
;;;  restrictions                                             ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule Continue-Mission_unrestricted
  (logical    (Equipment_Status normal)
              (Maneuvering_Status unrestricted)
              (Environment_Status normal)
              (Navigation_Status within_tolerance)
              (Spec_Mission_Status feasible))
    =>
   (decision-change Overall_Mission Continue-Mission_unrestricted
            High  Continue-mission-with-no-restrictions)
  (assert (Overall_mission_status Continue_Unrestricted)))



 ;;;;;;;;;;;;;;;;;; Restricted Status Update ;;;;;;;;;;;;;;;;;;
 ;;;   If the recovery evaluation is poor (as determined by ;;;
 ;;; by exceeding a standard recovery time) then abort the  ;;;
 ;;; mission .                                              ;;;
 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule Continue-mission_restricted-update
  (Overall_mission_status Continue_with_Restrictions)
  (recovery_evaluation poor)
 =>
 (decision-change Overall_Mission Continue-mission_restricted-update
           Assessment Abort_Mission)
  (assert (Overall_mission_status Abort_Mission)))
```

```
;;;;;;;;;;;;;; Initial Restricted Status Actions ;;;;;;;;;;;;;
;;;;  Note the mission_degradation status                    ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule  Continue_mission_restricted_initial
   (Overall_mission_status Continue_with_Restrictions)

  =>
   (decision-change Overall_Mission
                    Continue_mission_restricted_initial
                    Assessment Note-time-of-status-change)
   (bind ?*mission_degradation_time* (- (time) ?*start_time*)))



;;;;;;;;;;;;;;;;;;  Abort Mission  ;;;;;;;;;;;;;;;;;;;;;;;;;;
;;   A mission abort causes the overall mission status to ;;
;;   be locked.   A replan must be made to reach the      ;;
;;   abort rendezvous                                     ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

  ;;;; Default is that AUV can return under own power after
 ;;;; a mission abort.  However, if there is a primary control
 ;;;; system failure such as failure of rudders or dive-planes,
 ;;;; the vehicle will require recovery.


(defrule Abort_Mission
  (declare (salience 500))
        (Overall_mission_status Abort_mission)
    ?change <- (propagate-change down)
    ?point  <- (waypoint ?no)
   =>
   (retract ?point)
  (retract ?change)
  (decision-change Overall_Mission Abort_Mission Low
          lock_status_and_replan_route_to_abort_rendezvous)

   (undefrule  Overall_Mission_Assessor)              ;  status lock
   (do-for-instance ((?control CONTROL_SYSTEM))
                      (and (eq ?control:status INOPERATIVE)
                           (neq ?control Hover-Thrusters))

 (progn (Call-Guidance-Command turn_on_transponder transponder)
        (Call-Guidance-Command ascend_surface  planes)
  (printout t crlf crlf ">>>>> Shutting Down for Dynamic Recovery
                                                  <<<" crlf
                        ">>>>> Transponder will function for 2 hours
                                                 <<<" crlf)
  (halt)))
    (Abort-Route ))
```

```
;;;
```

```
    (printout t    "======================================" crlf
                   "|          EQUIPMENT DOWN             |" crlf
                   "--------------------------------------"  crlf)

   (do-for-all-instances ((?equipment SYSTEM_MONITOR))
                         (eq ?equipment:status INOPERATIVE)
      (printout t ">>>>>>>>  " (instance-name-to-symbol ?equipment)
                  "<<<<<<<<  "  crlf)))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                          Show Status Board                            ;;
;;;                                                                       ;;
;;; Shows the status of vehicle worlds and actions being taken            ;;
;;; to offset deviations or  discrepancies.  Not as timely as             ;;
;;; propagation flow of decisions, this only shows the effects            ;;
;;; of a decision since the last low level command to Guidance            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule show_status_board
      (Overall_mission_status ?status)
      (Maneuvering_Status ?maneuvering)
      (Navigation_Status  ?navigation)
      (Environmental_Status ?environment)
      (Equipment_Status ?equipment)
      (Spec_Mission_Status ?mission)
      (waypoint ?number)
      ?current <- (Current_action ?action)
      ?show <-(show board)
    =>
    (do-for-instance ((?point POSTURE))
                      (eq ?point:number (- ?number 1 ))
    (progn  (bind ?depth-configuration  ?point:action)
            (bind ?configuration        ?point:configuration)))
    (display-status  ?number ?status ?maneuvering ?navigation
                    ?environment ?equipment mission ?action
                        ?depth-configuration ?configuration)
    (retract ?current)
    (retract ?show))



;;;;;;;;;;;;;;;;;;;;;;;;  End of Main Module ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;    Programmer              :    W P Wilkinson
;    System                  :    CLIPS 5.0
;    Program                 :    AUV Mission Executor "SKIPPER"
;    Functional Area         :    Navigation
;    Latest Revision         :    30 August 91
;;;
```

```
      else
        (if (or (= ?*QtyNavProblems* 2)
                (= ?*NrNavInstrumentsfailed* 2)) then
         (retract ?nav)
         (assert (Navigation_Status critical))
         (assert (propagate change)))))



;;;;;;;;;;;    Separate Equipment Consideration

(defrule Navigation_Assessment_Equipment
   (declare (salience ?*navigation_salience*))
     (Equipment_Failure NAVIGATION_INSTRUMENT ?instrument)
     ?nav <- (Navigation_Status ?navstatus)
     =>
     (decision-change Navigation Navigation_Assessment Assessment
           determine_Nav_Status_and_pass_to_Overall_Mission_assessor)
     (bind ?*QtyNavProblems* (+ ?*QtyNavProblems* 1))
        (if (or (>= ?*QtyNavProblems* 4)
                (> ?*NrNavInstrumentsfailed* 2)) then
            (retract ?nav)
            (assert (Navigation_Status out_of_tolerance))
            (assert (propagate change))
      else
        (if (or (= ?*QtyNavProblems* 2)
                (= ?*NrNavInstrumentsfailed* 2)) then
          (retract ?nav)
          (assert (Navigation_Status critical))
          (assert (propagate change)))))




;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;; Although the AUV's propulsion power source is good for
;;;;;;;; approximately  2 hour mission, even long testing facility
;;;;;;;;  missions may cause an abort.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule Energy_Assessment
   (Energy_Deviation major)
    ?status <-(Navigation_Status ?navstatus)
    =>
    (retract ?status)
    (assert (Navigation_Status out_of_tolerance))
    (assert (propagate change)))
```

```
;;;
```

```
                        put-action no-depth-change))
            (if (and (> ?current:z_pos ?next:z_pos)
                     (neq ?next:z_pos 0.0)) then
                        (send (symbol-to-instance-name ?current)
                                    put-action ascent))
           (if (and (> ?current:z_pos ?next:z_pos)
                    (eq ?next:z_pos  0.0)) then
                         (send (symbol-to-instance-name ?current)
                        put-action surface))
           (if (< ?current:z_pos ?next:z_pos ) then
                      (send (symbol-to-instance-name ?current)
                                      put-action dive))))
              (Call-Guidance-Command mark_on_top waypoint)
              (assert (delta_depth_check complete))
              (assert (time-distance-check)))



  (defrule Waypoint_monitor
     ?point <- (waypoint ?no)
     ?depth-check <- (delta_depth_check complete)
     (configuration ?config)
   =>
    (decision-change Navigation Waypoint_monitor Low_assessment
                    assess_next_waypoint_and_sequence)
    (bind ?next_point (+ ?no 1))
    (retract ?point)
    (do-for-instance ((?destination POSTURE ))
                      (eq ?destination:number ?next_point)
             (Call-Guidance-Waypoint ?destination))
    (retract ?depth-check)
    (assert (waypoint ?next_point)))



   ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
   ;;;   Performs a <<time-distance>> check if passing a waypoint;
   ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule Waypoint_DistanceTimeEnergy_Check
   (waypoint ?no)
   ?t-check <-(time-distance-check)
  =>
  (decision-change Navigation Waypoint_DistanceTime_Check
            Low_assessment  determine_if_need_to_increase_speed)
  (bind ?energydepletion  (* .00013  ?*mission_time*))
           (if (> ?energydepletion .70) then
               (assert (Energy_Deviation major)))
           (do-for-instance ((?point POSTURE))
                       (eq ?point:number ?no)
```

```
            (if (> (abs (- ?*mission_time* ?point:ETA)) 40.0) then
                   (assert (Time_Deviation))
            else
                (if (> (abs (- ?*mission_time* ?point:ETA)) 20.0) then
      (Call-Guidance-Command Increase-Speed Drive_motors))))
                       (retract ?t-check))


   (defrule Time_Deviation
      (Time Deviation)
     =>
    (Replan-Route none ?*Goalx* ?*Goaly*  ?*Goalz*))



;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                                                                          ;;;
;;;                          Depth  Rules                                    ;;;
;;;  These rules require a direct depth-check from sonar. Currently          ;;;
;;;   exceptions to correct bottom following are signalled by the            ;;;
;;;   emergency  obstacle flag                                               ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


   (defrule depth_sounding_deviation_short_range
     (or (obstacle-flag-emergency  0001)
         (obstacle-flag-emergency  0011)
         (obstacle-flag-emergency  0101)
         (obstacle-flag-emergency  0111)
         (obstacle-flag-emergency  1101)
         (obstacle-flag-emergency  1011))
     =>
        (decision-change Navigation depth_sounding_deviation_shortrng
                         low_supervisory avoid_possible_shoaling)
        (Call-Guidance-Command ascend-?*safe_depth* planes)
        (bind ?*NrBottomObstacles*  (+ ?*NrBottomObstacles* 1))
        (assert (Depth-Status Violation)))

     ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
     ;;;;    Depth Sounding deviation at the limit of sound sensors ;;;
     ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


   (defrule depth_sounding_deviation_long_range
      (obstacle_alert on)
      (new_obstacle on)
     =>
      (decision-change Navigation depthsounding_deviation_longrng
                       low_supervisory avoid_possible_shoaling_early)
      (do-for-instance ((?obstacle OBSTACLE))
```

```
                    (and  (eq ?obstacle:type bottom)
                          (eq ?obstacle:ID_num ?*obstacle_ref*))
     (progn
          (bind ?*NrBottomObstacles*  (+ ?*NrBottomObstacles* 1))
          (Call-Guidance-Command ascend-?*safe_depth* planes)
          (assert (Depth-Status Violation)))))


;;;;  Aggregate of obstacles over short period of time indicates the AUV
;;;;  is in a serious potential grounding situation

  (defrule Detect_Shoaling
     (Depth-Status Violation)
     (test (> ?*NrBottomObstacles* 1))
     =>
     (decision-change Navigation Detect_Shoaling Low_assessment
                  determine_if_really_shoaling_or_just_bottom_obstcl)
     (if (and (> ?*NrBottomObstacles*  4)
              (< (- ?*mission_time* ?*BottomObstacleTime*)
                           ?*bottom_obstacle_time_interval*)) then
          (assert (Depth-Violation Shoaling))
          (Call-Guidance-Command Stop Drive-motors))
        (bind ?*BottomObstacleTime* ?*mission_time*))
```

```
;  Programmer         :    W P Wilkinson
;  System             :    CLIPS 5.0
;  Program            :    AUV Mission Executor "SKIPPER"
;  Functional Area    :    Maneuvering Rules
;  Latest Revision    :    30 August 91
;;;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;                                                                  ;;;;
;;;;;              Maneuvering Status Assessment                       ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;; This is the functional area supervisor for maneuvering.;;

       (defrule Maneuvering_Status_Assessment
        (declare (salience ?*maneuver_salience*))
         ?obst <-    (Obstacle_Avoidance restricted)
         ?assess <-  (Maneuvering-Status-Assess)
         ?maneuver <- (Maneuvering_Status ?status)
       =>
       (decision-change Maneuvering Maneuvering_Status_Assessment
              maneuvering-assessment  change-overall-maneuvering-status)
       (bind ?*maneuverability_factor* (+ ?*maneuverability_factor* 1))
         (if   (> ?*maneuverability_factor* 2)  then
             (retract ?maneuver)
             (retract ?obst)
             (assert (Maneuvering_Status severely_restricted))
             (assert (propagate change))
           else
            (retract ?maneuver)
            (retract ?obst)
            (assert  (Maneuvering_Status restricted))
            (assert  (propagate change)))
            (retract ?assess))



(defrule Maneuvering_Status_Assessment_long_range
       (declare (salience ?*maneuver_salience*))

       ?m_ability <- (maneuvering_ability ?ability)
       ?assess <-  (Maneuvering-Status-Assess)
       ?maneuver <- (Maneuvering_Status ?status)
       =>
       (decision-change Maneuvering Maneuvering_Status_Assessment
              maneuvering-assessment  change-overall-maneuvering-status)
       (bind ?*maneuverability_factor* (+ ?*maneuverability_factor* 1))
         (if (or (> ?*maneuverability_factor* 2)
                 (eq ?ability Major_Restriction)) then
             (retract ?maneuver)
             (assert (Maneuvering_Status severely_restricted))
             (assert (propagate change))
           else
             (retract ?maneuver)
            (assert  (Maneuvering_Status restricted))
            (assert  (propagate change)))
             (retract ?assess))
```

```
    (defrule Maneuvering_Equipment_Failure
    (Equipment_Failure Control_System ?control&:(neq ?control Hover-
                                                    Thrusters))

    =>
     (decision-change Maneuvering Maneuvering_Status_Assessment
              maneuvering-assessment  change-overall-maneuvering-status)
     (assert (Maneuvering_Status severely_restricted))
     (assert (propagate change)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;  Emergency Evasive Maneuvers for Obstacles at Close Range  ;;;
;;;;  Based upon obstacle alert system developed by C. FLOYD    ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

      (defrule emergency_maneuver_evaluation

        (or (obstacle-flag-emergency ?)
            (new-obstacle on))
        =>
        (decision-change Maneuvering emergency_maneuver_evaluation
                         assessment
                        assess_emergency_obstacle_avoidance_maneuvers )
        (bind ?*avoidance_time* ?*mission_time*)
        (assert (assess_avoidance_maneuver)))

      (defrule Assess_Avoidance_Maneuver
        (declare (salience -500))
        ?assess <- (assess_avoidance_maneuver)
        ?check <- (avoidance_time_check)
        =>
        (retract ?check)
       (if (> ?*mission_time* (+ ?*avoidance_time* ?*recovery_time*))then
        (retract ?assess)
        (assert (maneuvering_ability Major_Restriction))
        (assert (Maneuvering-Status-Assess))))



      (defrule emergency-evasive-maneuver-ascend
        (declare (salience 1000))
        (or (obstacle-flag-emergency 0001)
            (obstacle-flag-emergency 0011)
            (obstacle-flag-emergency 0101)
            (obstacle-flag-emergency 0111))
      =>
         (decision-change Maneuvering emergency-evasive-maneuver-ascend
```

```
        Low-supervisory-level ascend_to_avoid_obstacle )
   (Call-Guidance-Command  ascend-?*safe_depth*  rudder )
   (assert (Obstacle_Avoidance restricted)))



(defrule emergency-evasive-maneuver-leftascend
   (declare (salience 1000))
   (obstacle-flag-emergency 1101)
  =>
  (decision-change Maneuvering emergency_evasive_maneuver-leftascend
             Low-supervisory-level
             turn_left_and_ascend_to_avoid_obstcl)
   (Call-Guidance-Command  turn-left rudder)
   (Call-Guidance-Command  ascend-10    planes)
   (assert (Obstacle_Avoidance restricted)))


 (defrule emergency-evasive-maneuver-left
   (declare (salience 1000))
   (obstacle-flag-emergency 1100 )
    =>
   (decision-change Maneuvering emergency-evasive-maneuver-left
            Low-supervisory-level turn_left_to_avoid_obstacle)
   (Call-Guidance-Command  turn-left rudder)
   (assert (Obstacle_Avoidance restricted)))


 (defrule emergency-evasive-maneuver-rightascend
   (declare (salience 1000))
   (obstacle-flag-emergency 1011 )
    =>
   (decision-change Maneuvering emergency-evasive-maneuver-
                                        rightascend
                          Low-supervisory-level
                          turn_right_and_ascend_to_avoid_obstacle)
   (Call-Guidance-Command  turn-right rudder)
   (Call-Guidance-Command  ascend     planes)
   (assert (Obstacle_Avoidance restricted)))


 (defrule emergency-evasive-maneuver-right
   (declare (salience 1000))
   (obstacle-flag-emergency 1010)
   =>
   (decision-change Maneuvering emergency-evasive-maneuver-right
          Low-level-supervisory  turn_right_to_avoid_obstacle)
   (Call-Guidance-Command  turn-right rudder)
   (assert (Obstacle_Avoidance restricted)))
```

```
(defrule emergency-evasive-maneuver-stopascend
   (declare (salience 1000))
   (or (obstacle-flag-emergency 1110)
       (obstacle-flag-emergency 1111))
=>
  (decision-change Maneuvering emergency-evasive-maneuver-stopascend
            Low-level-supervisory Stop_forward_movement_and_ascend)
   (Call-Guidance-Command  Stop Drive-motors)
   (Call-Guidance-Command  ascend  planes)
   (assert (Obstacle_Avoidance restricted)))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;   Special configurations which can easily become                    ;;;;
;;;;   catastrophic if an abnormal condition exists.                     ;;;;
;;;;   Diving,  ascending and surfacing require                          ;;;;
;;;;   fast reaction to counter an unstable control                      ;;;;
;;;;   system or an obstacle close-aboard                                ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

   (defrule abnormal_surface
    (configuration ?config)
    (action   surface)
    (or (Equipment_Failure Control_System Plane_Controls)
        (obstacle_clearance ?clearance&:(neq ?clearance normal)))

     =>
    (decision-change Maneuvering abnormal_surface
                Low-supervisory-level
                Increase_speed_to_surface)
    (Call-Guidance-Command Increase-Speed Drive-motors)
    (assert (maneuvering_ability Major_Restriction))
    (assert (Assess-Manueuvering-Status)))




   (defrule abnormal_ascent
    (configuration ?config)
    (action   surface)
    (or (Equipment_ Failure Control_System Plane_Controls)
         (obstacle_clearance ?clearance&:(neq ?clearance normal)))
     =>
   (decision-change Maneuvering abnormal_ascent Low_supervisory_level
              increase_speed_of_ascent)
    (Call-Guidance-Command Increase-Speed Drive-motors)
    (assert (maneuvering_ability Major_Restriction))
    (assert (Maneuvering-Status-Assess)))
```

```
(defrule abnormal_dive
 (configuration ?config)
 (action  dive)
 (or (Equipment_Failure Control_System Plane_Controls)
     (obstacle_clearance ?clearance&:(neq ?clearance normal)))
   =>
(decision-change Maneuvering abnormal_dive  Low_supervisory_level
          decrease_speed_of_dive_ascend_to_safe_depth)
 (Call-Guidance-Command  Decrease-Speed Drive-motors)
 (Call-Guidance-Command  ascend-?*safe_depth* Planes)
 (assert (maneuvering_ability  Major_Restriction))
 (assert (Maneuvering-Status-Assess)))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                    Sensor_Limit Obstacle Detection                    ;;;
;;;                                                                       ;;;
;;;                                                                       ;;;
;;;    These Rules interface with the Obstacle Avoidance                  ;;;
;;;    DecisionMaker.  Most of these conditions can only be               ;;;
;;;    simulated until the Obstacle Avoidance DecisionMaker               ;;;
;;;    is completed                                                       ;;;
;;;                                                                       ;;;
 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


   ;;; Detection of a "new" obstacle ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
   ;;; The obstacle is assigned an ID reference number for tracking ;;
   ;;; As the OBSTACLE class message-handler indicates above,       ;;
   ;;; we are only interested in obstacles in a 180-degree arc      ;;
   ;;; about the bow                                                ;;
   ;;; the bow.                                                     ;;
   ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


      (defrule Obstacle_Detection_Normal_Limits
      ?obstflag <- (obstacle_alert on)
      ?new_one  <- (new_obstacle on)
      =>
        (decision-change Maneuvering Obstacle_detection_Normal_Limits
                 Low_assessment  classify_normal_range_obstacle_as_new)
       (bind ?*obstacle_ref* (+ ?*obstacle_ref* 1))
       (make-instance (gensym*) of OBSTACLE
                              (ID_num        (read obstacles))
                              (bearing       (read obstacles))
                              (type          (read obstacles))
                              (proximity     (read obstacles))
```

```
                                (brng_drift     (read obstacles))
                                (time_observed (read obstacles))
                                (confidence_factor (read obstacles))
                                (collision_danger unknown))
          (do-for-instance ((?obstacle OBSTACLE))
                           (eq ?obstacle:ID_num ?*obstacle_ref*)
                (send ?obstacle obstacle-change))
          (retract ?new_one))

    ;;; Update to previously detected obstacle

       (defrule Obstacle_Update
        ?obstflag <- (obstacle_alert on)
        ?update   <- (obstacle_update on)
        =>
        (decision-change Maneuvering Obstacle_Update Low_assessment
                   update_obstacle_status:rangebearing,collision-danger)
        (bind ?current_obstacle (read obstacles))
        (do-for-instance ((?obstacle OBSTACLE))
                         (eq ?obstacle:ID_num ?current_obstacle)
       (progn (send ?obstacle put-bearing (read obstacles))
              (send ?obstacle put-type    (read obstacles))
              (send ?obstacle put-proximity (read obstacles))
              (send ?obstacle put-brng_drift (read obstacles))
              (send ?obstacle put-time_observed  (read obstacles))
              (send ?obstacle put-confidence_factor (read obstacles))
              (send ?obstacle put-collision_danger unknown)
              (send ?obstacle obstacle-change)))
        (retract ?update))



  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  ;;; Determines whether proportional amount of obstacles are to the;;;
  ;;;left or right to heuristically determine which way to turn. If ;;;
  ;;;equally blocked on both sides, calls for a replan of the route.;;;
  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

    (defrule Collective_Obstacle_Assessment
      (collective_obstacle_assessment)
    =>
     (decision-change Maneuvering Collective_Obstacle_Assessment
        Low_assessment
assess_whether_presents_a_collision_danger_and_turn)
     (bind ?obstacles_left 0)
     (bind ?obstacles_right 0)
     (do-for-all-instances ((?obstacle OBSTACLE))
          (eq ?obstacle:collision_danger YES)
       (if (and (>= ?obstacle:bearing 270.) (<= ?obstacle:bearing 359.))
then
```

```
     (bind ?obstacles_left (+ ?obstacles_left 1))
     else
     (bind ?obstacles_right (+ ?obstacles_right 1))))
(if (> ?obstacles_left  ?obstacles_right) then
     (bind ?turn turn_right))
(if (> ?obstacles_right ?obstacles_left) then
     (bind ?turn turn_left))
(if (and (eq ?obstacles_right ?obstacles_left)
         (neq ?obstacles_right 0)) then
     (bind ?turn reverse_course)
      (assert (Obstacle_Avoidance restricted))
     (Replan-Route ?turn ?*Goalx* ?*Goaly* ?*Goalz*))
(if (or (> ?obstacles_left 0) (> ?obstacles_right 0)) then
  (assert (Call-Guidance-Command  ?turn rudder)))
  (assert (Maneuvering-Status-Assess)))
```

```
   Programmer          :   W  P  Wilkinson
;  System              :   CLIPS 5.0
;  Program             :   AUV Mission Executor  "SKIPPER"
;  Functional Area     :   Sub_System Monitoring  (Vehicle Internal World)
;  Latest Revision     :   30 August 91
; ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                              Description
;   This is the high-level abstraction of the system monitoring
;   functions of the AUV. This module is designed to be an overall
;   subsytem "health" monitor and performs both high-level and low
;   level polling of subsystems state. Some of the AUV systems
;   (all of which are modeled as objects) include the
;   power_sources, navigation instruments, sonars, environmental_sensors
;   and control_systems such as rudders, planes, thrusters.  A continuous
;   loop polls all systems, and the message-handlers associated with each
;   class attempt to determine if a reading is out of range. These in
;   turn produce facts which cause equipment rules to fire.
;   Failure conditions cause the Equipment assessor rule to determine
;   if an equipment going critical or an equipment failure will cause
;   a restriction. If the functional area of Equipment_Status has a
;   a degradation, this is passed to the Overall Mission Assessor in
;   in main file skipper.clp .
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;   Global Variables Pertaining to Equipment Monitoring;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

         (defglobal    ?*sysmonitor_salience*            =   100
                       ?*QtyEquipment_failed*            =   0
                       ?*NrNavInstrumentsfailed*         =   0
                       ?*NrSonarfailed*                  =   0
                       ?*NrEnvironSensorsfailed*         =   0)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;               AUV  Subsystem  Monitor Objects                ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

   (defclass SYSTEM_MONITOR (is-a USER)
      (slot  type_of_reading)
      (slot  reading)
      (slot  degradation_time)
      (slot  status (default NORMAL))
      (slot  Redundant_Equipment (initialize-only))
      (slot  redline_high (initialize-only))
      (slot  guardline_high (initialize-only))
      (slot  guardline_low  (initialize-only))
      (slot  redline_low    (initialize-only)) )
```

```
;;;;   Power sources which support the various equipments ;;;;


(defclass POWER_SOURCE (is-a SYSTEM_MONITOR)
   (slot  type_of_reading      (default power_in_watts))
   (slot  Redundant_Equipment (default  NONE))
   (slot  Alternate_Source     (initialize-only))
   (slot  redline_high         (default 0.0))
   (slot  guardline_high       (default 0.0))
   (slot  Equipment_Supported)
   (message-handler get-reading))


(defmessage-handler POWER_SOURCE get-reading after ()
    (if (and (< ?self:reading ?self:guardline_low)
             (> ?self:reading ?self:redline_low)) then
     (assert (Equipment_Critical ?self:Equipment_Supported))
     (assert (Power_Source failure ))))


;;;;;;;   Sonar class and objects  ;;;;;;;;;;

(defclass SONAR (is-a SYSTEM_MONITOR)
   (slot type-of-reading (default frequency_in_hz))
   (slot redline_high    (default 50.0))
   (slot guardline_high  (default 40.0))
   (slot guardline_low   (default 5.0))
   (slot redline_low     (default 1.0))
   (slot statuschange_time   (default 0.0))
   (slot recovery_time   (default 20.0))
   (message-handler  get-reading))


;;;;;;;;;;   Check Sonar readings for out-of-limit readings ;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defmessage-handler SONAR get-reading after ()
   (bind ?sonar (instance-name-to-symbol (instance-name ?self)))
   (if (or (and (> ?self:reading ?self:guardline_high)
                (< ?self:reading ?self:redline_high))
           (and (< ?self:reading ?self:guardline_low)
                (> ?self:reading ?self:redlinew))) then
    (assert (Equipment_Critical Sonar   ?sonar))
    (send ?self put-statuschange_time ?*mission_time*)
    (send ?self put-status CRITICAL)
   else
     (if (or (> ?self:reading ?self:redline_high)
             (< ?self:reading ?self:redline_low)) then
          (assert (Equipment_Failure Sonar   ?sonar))
          (bind ?*NrSonarfailed* (+ ?*NrSonarfailed* 1))
   (send ?self put-status INOPERATIVE))))
```

```
;;;;;;;;;;;;;;;;    Navigation Instruments  ;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defclass NAVIGATION_INSTRUMENT (is-a SYSTEM_MONITOR)
   (slot type_of_reading (default power_in_watts))
   (slot time_critical   (default 0.0))
   (message-handler get-reading ))


 ;;;;;;;;;;;;;;;;    If a Navigation instrument is out of limits
 ;;;;;;;;;;;;;;;;  then tabulate the number failed and declare it
 ;;;;;;;;;;;;;;;;  failed

 (defmessage-handler NAVIGATION_INSTRUMENT get-reading after ()
  (bind ?instrument (instance-name-to-symbol (instance-name ?self)))
    (if  (or (> ?self:reading ?self:guardline_high)
                (< ?self:reading ?self:guardline_low)) then
     (assert (Equipment_Failure Nav_Instrument ?instrument))
   (bind ?*NrNavInstrumentsfailed* (+ ?*NrNavInstrumentsfailed* 1))
     (send ?self put-status INOPERATIVE)))

;;;;;;;;;;;;;;;;     Control  Systems   ;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;; If these fail, this will eventually cause a mission ;
;;;;;;;;;;;; abort, unless the control is a Hover-Thruster, which;
;;;;;;;;;;;; at this stage of AUV development, is not mission-   ;
;;;;;;;;;;;; critical                                            ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defclass CONTROL_SYSTEM (is-a SYSTEM_MONITOR)
   (slot type_of_reading (default potential_in_volts))
   (slot statuschange_time   (default 0.0))
   (slot recovery_time       (default 10.0))
   (slot control-type)
   (slot response (default normal))
   (message-handler get-reading)
   (message-handler get-response))


(defmessage-handler CONTROL_SYSTEM get-reading after ()
   (bind ?control (instance-name-to-symbol (instance-name ?self)))
   (if (or (and (> ?self:reading ?self:guardline_high)
                (< ?self:reading ?self:redline_high))
           (and (< ?self:reading ?self:guardline_low)
                (> ?self:reading ?self:redline_low))) then
    (assert (Equipment_Critical Control_System ?control))
    (send ?self put-status CRITICAL)
    (send ?self put-statuschange_time ?*mission_time*)
   else
       (if (or (> ?self:reading ?self:redline_high)
```

```
              (< ?self:reading ?self:redline_low)) then
          (assert (Equipment_Failure Control_System ?control))
          (send ?self put-status INOPERATIVE))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;; This is an added check for control systems instead of the
;;;; just the  electrical status. If a control system does not
;;;; respond or is in the wrong position , this is an indication
;;;; of impending failure and is justification for a status of
;;;; CRITICAL
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defmessage-handler CONTROL_SYSTEM get-response after ()
   (if (neq ?self:response normal)  then
      (assert (Equipment_Critical Control_System ?self))
      (send ?self put-status CRITICAL))
      (send ?self put-statuschange_time ?*mission_time*))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;   Environmental Sensors are evaluated for both
;;; electrical status and the environmental reading they
;;; indicate even when operating properly.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defclass ENVIRON_SENSORS (is-a SYSTEM_MONITOR)
   (slot  type  (initialize-only))
   (slot  environmental_reading )
   (slot  environment_upperlimit (initialize-only))
   (slot statuschange_time   (default 0.0))
   (slot  Redundant_Equipment (default NONE))
   (message-handler get-reading))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;    This checks the environmental sensors for proper    ;;;
;;;    operation                                           ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

 (defmessage-handler ENVIRON_SENSORS get-reading after ()
    (bind  ?sensor (instance-name-to-symbol (instance-name ?self)))
    (if (or (> ?self:reading ?self:redline_high)
                  (< ?self:reading ?self:redline_low)) then
      (assert  ( Equipment_Failure Environ_Sensor ?sensor ))
    (bind ?*NrEnvironSensorsfailed* (+ ?*NrEnvironSensorsfailed* 1))
      (send ?self put-status INOPERATIVE)))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;                  Environment  Limits  Readings                    ;;;
;;;;     This checks the environmental sensors for                    ;;;
;;;;     environmental conditions which are out of limits.            ;;;
;;;;     The rules which operate on these limits  are part            ;;;
;;;;     of the environmental world and are found in module           ;;;
;;;;    environment.clp.  This message-handler operates               ;;;
;;;;    on that world and   is only included here for                 ;;;
;;;;    convenience and polling.                                      ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

 (defmessage-handler ENVIRON_SENSORS get-environmental_reading
                                                      after ()
    (bind ?sensor (instance-name-to-symbol (instance-name ?self)))
   (if  (> ?self:environmental_reading ?self:environment_upperlimit)
     then
       (assert (Adverse_condition ?self:type ?sensor))
       (send ?self put-status INOPERATIVE)))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;  Mission equipment is represented among the equipment              ;;;
;;;; monitoring objects although it has little bearing on              ;;;
;;;; present AUV missions in the NPS pool.                             ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

 (defclass MISSION_EQUIPMENT (is-a SYSTEM_MONITOR)
    (slot  type_of_reading (default potential_in_volts))
    (slot statuschange_time   (default 0.0))
    (slot  type (initialize-only))
    (message-handler get-reading))


  (defmessage-handler MISSION_EQUIPMENT get-reading after ()
   (bind ?instrument (instance-name-to-symbol (instance-name ?self)))
     (if  (or (> ?self:reading ?self:guardline_high)
              (< ?self:reading ?self:guardline_low)) then
     (assert (Equipment_Failure Mission_Instrument ?instrument))
     (send ?self put-status INOPERATIVE)))
```

```
;;;
```

```
(Aux2_Bat  of POWER_SOURCE          (reading   25.0)
                                    (Alternate_Source NONE)
                                    (guardline_low  7.0)
                                    (redline_low    1.0)
                                (Equipment_Supported None NONE))

(DeadReckon_Bat of POWER_SOURCE     (reading   25.0)
                                    (Alternate_Source Aux2_Bat)
                                    (guardline_low  7.0)
                                    (redline_low    1.0)
     ( Equipment_Supported Navigation_Instrument
                                 DeadReckonAnalyzer ))

(Gyro_Bat  of POWER_SOURCE          (reading 25.0)
                                    (Alternate_Source Aux2_Bat)
                                    (guardline_low  7.0)
                                    (redline_low    1.0)
       (Equipment_Supported Navigation_Instrument Gyro))

(DeadReckonAnalyzer of NAVIGATION_INSTRUMENT (reading     5.0)
                                    (Redundant_Equipment Gyro )
                                           (redline_high 10.0)
                                           (guardline_high 8.0)
                                           (guardline_low  4.0)
                                          (redline_low    2.0))

(Gyro of NAVIGATION_INSTRUMENT     (reading  4.0)
                                   (Redundant_Equipment
                                      DeadReckonAnalyzer)
                                   (redline_high 8.0)
                                   (guardline_high 6.0)
                                   (guardline_low 2.0)
                                   (redline_low 1.5))

(Aux3_Bat  of POWER_SOURCE         (reading  50.0)
                                   (Alternate_Source NONE)
                                   (guardline_low  10.0)
                                   (redline_low    1.0)
                                   (Equipment_Supported NONE))


(Hover_Bat of POWER_SOURCE         (reading  50.0)
                                   (Alternate_Source Aux3_Bat)
                                   (guardline_low 10.0)
                                   (redline_low   1.0)
    (Equipment_Supported  Control_System Hover-Thrusters))

(Motor1_Bat of POWER_SOURCE        (reading  50.0)
                                   (Alternate_Source Aux3_Bat)
                                   (guardline_low 10.0)
                                   (redline_low   1.0)
```

```
            (Equipment_Supported Control_SystemDrive-Motor1))

(Motor2_Bat of POWER_SOURCE      (reading  50.0)
                                 (Alternate_Source Aux3_Bat)
                                 (guardline_low 10.0)
                                 (redline_low   1.0)
            (Equipment_Supported Control_System Drive-Motor2))

(Planes_Bat of POWER_SOURCE      (reading  50.0)
                                 (Alternate_Source Aux3_Bat)
                                 (guardline_low 10.0)
                                 (redline_low   1.0)
          (Equipment_Supported Control_System Plane-Controls))

(Rudder_Bat of POWER_SOURCE      (reading  50.0)
                                 (Alternate_Source Aux3_Bat)
                                 (guardline_low 10.0)
                                 (redline_low   1.0)
                               (Equipment_Supported Rudder))

(Hover-Thrusters of CONTROL_SYSTEM (reading 7.0)
                                (control-type auxiliary)
                                     (redline_high 10.0)
                                     (guardline_high 8.0)
                                     (guardline_low  4.0)
                                     (redline_low    2.0))
(Drive-Motor1 of CONTROL_SYSTEM
                                 (reading   7.0)
                                 (control-type propulsion)
                                 (redline_high 12.0)
                                 (guardline_high 8.0)
                                 (guardline_low  4.0)
                                 (redline_low    2.0))

(Drive-Motor2 of CONTROL_SYSTEM  (reading   7.0)
                                 (control-type propulsion)
                                 (redline_high 12.0)
                                 (guardline_high 8.0)
                                 (guardline_low  4.0)
                    (redline_low   2.0))
(Plane-Controls of CONTROL_SYSTEM (reading   5.0)
                                 (control-type depth)
                                 (redline_high 8.0)
                                 (guardline_high 6.0)
                                 (guardline_low  2.0)
                                 (redline_low    1.0))

(Rudder of CONTROL_SYSTEM        (reading   5.0)
                                 (control-type azimuth)
                                 (redline_high 8.0)
                                 (guardline_high 6.0)
```

```
                                    (guardline_low  2.0)
                                    (redline_low    1.0))

(Aux4_Bat of POWER_SOURCE           (reading  20.0)
                                    (Alternate_Source NONE)
                                    (guardline_low  10.0)
                                    (redline_low    1.0)
                                    (Equipment_Supported NONE))

(SeaTemp_Bat of POWER_SOURCE        (reading   20.0)
                                    (Alternate_Source Aux4_Bat)
                                    (guardline_low  5.0)
                                    (redline_low    1.0)
        (Equipment_Supported Environ_Sensor SeaTempSensor))

(SeaState_Bat of POWER_SOURCE        (reading   20.0)
                                     (Alternate_Source Aux4_Bat)
                                     (guardline_low  5.0)
                                     (redline_low    1.0)
       (Equipment_Supported Environ_Sensor SeaStateGyro))


(SeaTempSensor of ENVIRON_SENSORS  (reading        3.0)
                                  (environmental_reading 55.0 )
                                  (environment_upperlimit 90.0 )
                                     (type potential )
                                     (redline_high   5.0)
                                     (guardline_high 4.0)
                                     (guardline_low  1.0)
                                     (redline_low    0.5))

(SeaStateGyro of ENVIRON_SENSORS    (reading       5.0)
                                   (environmental_reading 1.0 )
                                   (environment_upperlimit 2.0 )
                                     (type potential_in_volts)
                                     (redline_high   8.0)
                                     (guardline_high 6.0)
                                     (guardline_low  2.0)
                                     (redline_low    1.0))

(PressureTransducer of ENVIRON_SENSORS
                                    (reading      50.0)
                                    (environmental_reading 50.0 )
                                  (environment_upperlimit 75.0 )
                                    (type  potential_in_volts)
                                      (redline_high   60.0)
                                      (guardline_high 55.0)
                                      (guardline_low  45.0)
                                      (redline_low    35.0))
(Hydrography_Instr1 of MISSION_EQUIPMENT
                                       (reading  3.0)
```

```
                                                (type      Surveying)
                                                (redline_high   5.0)
                                                (guardline_high 4.0)
                                                (guardline_low  1.0)
                                                (redline_low    0.5)))


;;;  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;       This continuously polls the equipment monitors to
;;;    determine if the equipment power readings are correct,
;;;    indicating that the equipment is functioning.
;;;  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;

  (defrule monitor_health_continuously
     (declare (salience -500))
     ?monitor <- (system_monitors running)
        =>

     (do-for-all-instances ((?sonar SONAR))
                           (neq ?sonar:status INOPERATIVE)
       (send ?sonar get-reading))
     (do-for-all-instances ((?power POWER_SOURCE))
                           (neq ?power:status INOPERATIVE)
       (send ?power get-reading))
     (do-for-all-instances ((?instrument NAVIGATION_INSTRUMENT))
                           (neq ?instrument:status INOPERATIVE)
       (send ?instrument get-reading))
     (do-for-all-instances ((?control CONTROL_SYSTEM))
                           (neq ?control:status INOPERATIVE)
       (send ?control get-reading))
     (do-for-all-instances ((?sensor ENVIRON_SENSORS))
                           (neq ?sensor:status INOPERATIVE)
       (progn (send ?sensor get-reading)
       (send ?sensor get-environmental_reading)))
    (do-for-all-instances ((?miss_instrument MISSION_EQUIPMENT))
                           (neq ?miss_instrument:status INOPERATIVE)
    (send ?miss_instrument get-reading))
    (retract ?monitor)
    (assert(check critical-equipment)))
```

```
(defrule check_critical_equipment
  (declare (salience -500))
   ?check <- (check critical-equipment)
   =>

   (do-for-all-instances ((?sonar SONAR))
         (eq ?sonar:status CRITICAL)
   (progn (if (and (> ?sonar:reading ?sonar:guardline_low)
                   (< ?sonar:reading ?sonar:guardline_high)) then
          (assert (Equipment_Recovery Sonar ?sonar))
          (put ?sonar:status NORMAL)
          else
     (if (> ?*mission_time* (+ ?sonar:statuschange_time
                               ?sonar:recovery_time)) then
            (put ?sonar:status INOPERATIVE)
            (assert (Equipment_Failure Sonar ?sonar))
            (send ?sonar put-status INOPERATIVE)))))

   (do-for-all-instances ((?control CONTROL_SYSTEM))
         (eq ?control:status CRITICAL)
   (progn (if (and (> ?control:reading ?control:guardline_low)
                   (< ?control:reading ?control:guardline_high)) then
            (assert (Equipment_Recovery Control_System ?control))
            (put ?control:status NORMAL)
         else
          (if (> ?*mission_time* (+ ?control:statuschange_time
                 ?control:recovery_time)) then
            (put ?control:status INOPERATIVE)
            (assert (Equipment_Failure Control_System ?control))
            (send ?control put-status INOPERATIVE)))))
   (retract ?check))



;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;            Assesses the impact of loss or crippling of     ;;;
;;;  vital equipment .                                         ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

   (defrule Equipment_Status_Assessment
     (declare (salience ?*sysmonitor_salience*))
     (or (Equipment_Critical ?class ?Equipment)
         (Equipment_Failure ?class ?Equipment))
         (Equipment_Mission_Essential ?essential)
     ?assessflag<- (Equipment-Status-Assess)
     ?statusflag <- (Equipment_Status ?status)
      =>
     (bind ?*sysmonitor_salience* (+ ?*sysmonitor_salience* 1))
     (decision-change System_Monitor Equipment_Status_Assessment
          Assessment  Assessing_Status )
```

```
    (bind ?*QtyEquipment_failed* (+ ?*QtyEquipment_failed* 1))
    (if (or (> ?*QtyEquipment_failed* 2)
            (eq ?essential yes)) then
       (retract ?statusflag)
       (assert (Equipment_Status  major_failure))
       (assert (propagate change))
    else
       (retract ?statusflag)
       (assert (Equipment_Status  equipment_critical))
       (assert (propagate change )))
       (retract ?assessflag))



;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;   Establishes whether or not equipment has recovered. ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule  Equipment_Recovery
  (declare (salience ?*sysmonitor_salience*))
  (Equipment_Recovery ?class ?equipment)
  =>
    (decision-change System_Monitor Equipment_Recovery Assessment
       resume_normal_equip_operations)
    (bind ?*QtyEquipment_failed* (- ?*QtyEquipment_failed* 1))
    (if (eq ?*QtyEquipment_failed* 0) then
       (assert (Equipment_Status normal))
       (assert (propagate change))))


;;;  attempts to shift to alternate power source if one avail
;;;  places failed power source in the INOPERATIVE mode

(defrule Power_Source_Critical
 (Power_Source failure)
 (Equipment_critical ?equip_class ?Equipment)
 (Alternate_Power_Source ?source)
=>
   (decision-change System_Monitor Power_Source_Critical Low
          shift-power-source )
  (Call-Guidance-Command shift-powersource-to ?source )
  (send (symbol-to-instance-name ?source)
             put-Equipment_Supported ?Equipment)
  (do-for-instance   ((?Battery POWER_SOURCE))
                        (eq ?Battery:Equipment_Supported ?Equipment)
 (send (symbol-to-instance-name ?Battery) put-status INOPERATIVE)))
```

```
;;; ;;;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;                  Control System Failure                          ;;;
;;;    Assesses any control system failure except for                ;;;
;;;    Hover-Thrusters as a failure of mission-essential             ;;;
;;;    (i.e., vital)  equipment.                                     ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule Control_System_Critical
 (Equipment_Critical Control_System ?control)
 =>
  (decision-change  System_Monitor Control_System_Critical Low
      Pass_info_to_Equip_Status_Assessor)
   (if (neq ?control Hover-Thrusters) then
   (assert (Equipment_Mission_Essential yes))
   (assert (Equipment-Status-Assess))))



 (defrule Control_System_Failure
   (Equipment_Failure Control_System ?control)
=>
   (decision-change System_Monitor Control_System_Failure  Low
             Pass_info_to_Equip_Assessor)
   (if (eq ?control Hover-Thrusters) then
       (assert (Equipment_Mission_Essential no))
    else
      (assert (Equipment_Mission_Essential yes))
      (assert (Equipment-Status-Assess))))


 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
 ;;;        Environmental  Sensor  Failure                          ;;;
 ;;;    These have the least effect on the Equipment               ;;;
 ;;;    functional area. The pressure transducer is the            ;;;
 ;;;    only environmental sensor considered vital                 ;;;
 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule Environmental_Sensor_Failure
 (Equipment_Failure Environmental_Sensor  ?sensor)
=>
   (decision-change  System_Monitor Environmental_Sensor_Failure
                 Low Pass_info_to_Equip_Status_Assessor)
    (if (neq ?sensor PressureTransducer) then
      (assert (Equipment_Mission_Essential no))
     else
   (assert (Equipment_Mission_Essential yes))
   (assert (Equipment-Status-Assess))))
```

```
;;;;  Programmer          :  W P Wilkinson
;;;;  System              :  CLIPS 5.0
;;;;  Program             :  AUV Mission Executor "SKIPPER"
;;;;  Functional Area     :  Environment
;;;;  Latest Revision     :  04 Sep 91
;;  ------------------------------------------------------------------
;;;;                          Description
;;;;     This is the abstraction of the Environmental world.
;;;;     Environmental out of limits readings cause the environment
;;;;     to degrade, but mostly are isolated phenomena. If a
;;;;     collective degradation occurs, this signifies a negative
;;;;     trend in the environment and reason for AUV to abort the
;;;;     mission.
;;;;   ------------------------------------------------------------------

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;  Global Variables Pertaining to Environment    ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


      (defglobal ?*environment_salience*   = 100
                 ?*QtyEnvironProblems*     =   0
                 ?*sea_state_thresh*       =   3)


 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
 ;;;                       Environmental Assessor                  ;;;
 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
 ;;;; This assumes that a single environment problem is not critical ;;;
 ;;;; in itself. Rather, an aggregate of out-of-range sensor readings  ;;;
 ;;;; indicate a large environmental phenomena such as a storm. In     ;;;
 ;;;; such a situation, the environmental  situation would be          ;;;
 ;;;; severely degraded, causing AUV to abort the mission.             ;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


   (defrule Environment_Assessor
      (declare (salience ?*environment_salience*))
       ?cond <- (Adverse_condition ?type ?equipment)
       ?current <-  (Environmental_Status ?status)
      =>
      (retract ?cond)
      (decision-change Environmental_world Environment_Assessor
                Assessment determine_if_environment_status_is_hazard)
      (bind ?*QtyEnvironProblems* ( + ?*QtyEnvironProblems* 1))
      (if (>= ?*QtyEnvironProblems*  3) then
         (retract ?current)
         (assert (Environmental_Status major_deviation))
         (assert (propagate change))))
```

```
;;;
```

```
(defrule temperature_sensor
   (Adverse_condition temperature ?equipment)
  =>
   (decision-change Environmental_world temperature_sensor Low_level
             determine_if_temp_change_indicates_navigation_error)
   (Call-Guidance-Command Verify-Location navigation))
```

APPENDIX B. TESTING SCENARIOS

1. SCENARIO 1

```
CLIPS> (watch statistics)
CLIPS> (batch upload.bat)
CLIPS> (close)
FALSE
CLIPS> (clear)
CLIPS> (load skipper.clp)
Defining defglobal: *start_time*
Defining defglobal: *mission_time*
Defining defglobal: *mission_degradation_time*
Defining defglobal: *recovery_time*
Defining defglobal: *Time_Interval*
Defining defglobal: *emergency_salience*
Defining defglobal: *mission_critical_power*
Defining defglobal: *Functional_area_failure*
Defining defglobal: *Functional_area_critical*
Defining defglobal: *current_event*
Defining defglobal: *Goalx*
Defining defglobal: *Goaly*
Defining defglobal: *Goalz*
Defining deffunction: show-demo-description
Defining deffunction: copy-old-instance
Defining defclass block DECISION
Defining deffunction: decision-change
Defining defclass block EVENT_SCHEDULE
Defining defmessage-handler execute-event primary in class EVENT_SCHEDULE.
Defining defclass block POSTURE
Defining deffunction: Call-Guidance-Waypoint
Defining deffunction: Call-Guidance-Command
Defining deffunction: Replan-Route
Defining deffunction: Abort-Route
Defining definstances block STARTING_DECISIONS
Defining deffacts: Starting_Facts
Defining defrule: initialize-vehicle +j
Defining defrule: upload +j+j+j+j
Defining defrule: Mission_Timer +j
Defining defrule: timer-manager +j
Defining defrule: Document_Mission +j
```

```
Defining defrule: event_schedule_manager +j
Defining deffunction: Total-Functional-Problems
Defining defrule: Overall_Mission_Assessor +j+j+j+j+j+j+j
Defining defrule: Continue-Mission_unrestricted +j+j+j+j+j
Defining defrule: Continue-mission_restricted-update +j+j
Defining defrule: Continue_mission_restricted_initial =j
Defining defrule: Abort_Mission +j+j+j
Defining deffunction: display-status
Defining defrule: show_status_board =j+j+j+j+j+j+j+j+j
CLIPS> (load maneuvering.clp)
Defining defglobal: *maneuver_salience*
Defining defglobal: *obstacle_ref*
Defining defglobal: *obstacle_clearance_time*
Defining defglobal: *avoidance_time*
Defining defglobal: *maneuverability_factor*
Defining defclass block OBSTACLE
Defining defmessage-handler obstacle-change primary in class OBSTACLE.
Defining defrule: Maneuvering_Status_Assessment +j+j+j
Defining defrule: Maneuvering_Status_Assessment_long_range +j+j+j
Defining defrule: Maneuvering_Equipment_Failure +j
Defining defrule: emergency_maneuver_evaluation +j
+j
Defining defrule: Assess_Avoidance_Maneuver +j+j
Defining defrule: emergency-evasive-maneuver-ascend +j
+j
+j
+j
Defining defrule: emergency-evasive-maneuver-leftascend +j
Defining defrule: emergency-evasive-maneuver-left +j
Defining defrule: emergency-evasive-maneuver-rightascend +j
Defining defrule: emergency-evasive-maneuver-right +j
Defining defrule: emergency-evasive-maneuver-stopascend +j
+j
Defining defrule: abnormal_surface +j+j+j
=j=j+j
Defining defrule: abnormal_ascent =j=j+j
=j=j=j
Defining defrule: abnormal_dive =j+j+j
=j=j+j
Defining defrule: Obstacle_Detection_Normal_Limits +j+j
Defining defrule: Obstacle_Update =j+j
Defining defrule: Collective_Obstacle_Assessment +j
CLIPS> (load navigation.clp)
```

```
Defining defglobal: *QtyNavProblems*
Defining defglobal: *NrNavInstrumentsfailed*
Defining defglobal: *NrBottomObstacles*
Defining defglobal: *navigation_salience*
Defining defglobal: *safe_depth*
Defining defglobal: *BottomObstacleTime*
Defining defglobal: *bottom_obstacle_time_interval*
Defining defrule: Navigation_Assessment +j+j
+j+j
Defining defrule: Navigation_Assessment_Equipment +j+j
Defining defrule: Energy_Assessment +j+j
Defining defrule: Goal_Recognition +j+j
Defining defrule: WaypointArrival-DepthComparison-GoalCheck +j+j+j
Defining defrule: Waypoint_monitor +j+j+j
Defining defrule: Waypoint_DistanceTimeEnergy_Check =j+j
Defining defrule: Time_Deviation +j
Defining defrule: depth_sounding_deviation_short_range =j
=j
=j
=j
=j
=j
Defining defrule: depth_sounding_deviation_long_range =j=j
Defining defrule: Detect_Shoaling +j
CLIPS> (load sensor.clp)
Defining defglobal: *sysmonitor_salience*
Defining defglobal: *QtyEquipment_failed*
Redefining defglobal: ?*NrNavInstrumentsfailed*
Defining defglobal: *NrSonarfailed*
Defining defglobal: *NrEnvironSensorsfailed*
Defining defclass block SYSTEM_MONITOR
Defining defclass block POWER_SOURCE
Defining defmessage-handler get-reading after in class POWER_SOURCE.
Defining defclass block SONAR
Defining defmessage-handler get-reading after in class SONAR.
Defining defclass block NAVIGATION_INSTRUMENT
Defining defmessage-handler get-reading after in class NAVIGATION_INSTRUMENT.
Defining defclass block CONTROL_SYSTEM
Defining defmessage-handler get-reading after in class CONTROL_SYSTEM.
Defining defmessage-handler get-response after in class CONTROL_SYSTEM.
Defining defclass block ENVIRON_SENSORS
Defining defmessage-handler get-reading after in class ENVIRON_SENSORS.
Defining defmessage-handler get-environmental_reading after in class
```

ENVIRON_SENSORS.
Defining defclass block MISSION_EQUIPMENT
Defining defmessage-handler get-reading after in class MISSION_EQUIPMENT.
Defining definstances block Sysmonitor-Bank
Defining defrule: monitor_health_continuously +j
Defining defrule: check_critical_equipment +j
Defining defrule: Equipment_Status_Assessment +j+j+j+j
+j+j+j+j
Defining defrule: Equipment_Recovery +j
Defining defrule: Power_Source_Critical +j+j+j
Defining defrule: Sonar_Failure +j
Defining defrule: Sonar_Critical +j
Defining defrule: Navigation_Instrument_Failure +j
Defining defrule: Control_System_Critical +j
Defining defrule: Control_System_Failure +j
Defining defrule: Environmental_Sensor_Failure +j
CLIPS> (load environment.clp)
Defining defglobal: *environment_salience*
Defining defglobal: *QtyEnvironProblems*
Defining defglobal: *sea_state_thresh*
Defining defrule: Environment_Assessor +j+j
Defining defrule: Environment_Assessor_Equipment +j+j
Defining defrule: attitude_sensor +j
Defining defrule: pressure_sensor +j
Defining defrule: temperature_sensor +j
CLIPS> (reset)
CLIPS> (run)

Welcome to the MISSION EXECUTOR DEMO

WAYPOINTS: All scenarios take place over the same set
of INITIAL waypoint coordinates.

EQUIPMENT: All equipment is simulated in the event file
Objects are created for each onboard equipment

SITUATIONS: All situations are also simulated in the event
file. For instance, an obstacle detection is

listed and this simulates the Obstacle Avoidance DecisionMaker passing this information through the interface to the Executor .

SCENARIO CHOICES: select number <Ret>
1 Waypoint_Hopping Only (transit)
2 Obstacle Avoidance
3 Vehicle Control System Failure
4 Obstacles and Environment Problems
5 Equipment Failures
6 Exit the Simulator

1

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Navigation
rule : WaypointArrival-DepthComparison
level : Low_assessment
action : determine_type_of_depth_change
time : 0.232

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Navigation
rule : Waypoint_DistanceTime_Check
level : Low_assessment
action : determine_if_need_to_increase_speed
time : 0.407

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Navigation
rule : Waypoint_monitor
level : Low_assessment
action : assess_next_waypoint_and_sequence
time : 0.569

==========
| Skipper's Display |
==========
TIME in min_secs 0:00
Overall Mission Status >>>> Continue_Unrestricted <<<<

```
 Manuevering_Status :  unrestricted
 Equipment_Status   :  normal
 Navigation_Status  :  within_tolerance
 Environment_status :  normal
 Spec_Mission_status:  feasible
-----------------------------------------------------|
| evolution     :  transit
| depth-status  :  dive
| =====================================================
| Last Command to Guidance :  underway
| enroute-waypoint         :  1
-------------------------------------------------------
|                Obstacles                  |
=============================================================
|  Direction  |    Proximity  | Type            |
-------------------------------------------------------
=========================================
|        EQUIPMENT DOWN           |
-----------------------------------

**********************************************
 Event Number :  1
_____________________________________________
 Description :  passing_waypoint
_____________________________________________
 Time       :   10.000
**********************************************

>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   :  Navigation
 rule   :  WaypointArrival-DepthComparison
 level  :  Low_assessment
 action :  determine_type_of_depth_change
 time   :   10.207


>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   :  Navigation
 rule   :  Waypoint_DistanceTime_Check
 level  :  Low_assessment
 action :  determine_if_need_to_increase_speed
 time   :   10.380
```

```
>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : Waypoint_monitor
 level  : Low_assessment
 action : assess_next_waypoint_and_sequence
 time   :   10.543

============================================================
|              Skipper's Display                 |
============================================================
  TIME in min_secs    0:10
  Overall Mission Status  >>>> Continue_Unrestricted <<<<
  Manuevering_Status :  unrestricted
  Equipment_Status   :  normal
  Navigation_Status  :  within_tolerance
  Environment_status :  normal
  Spec_Mission_status:  feasible
------------------------------------------------------------|
| evolution     : transit
| depth-status  : dive
| ==========================================================
| Last Command to Guidance :  mark_on_top
| enroute-waypoint         :  2
-------------------------------------------------------------
|                Obstacles                       |
============================================================
|  Direction   |    Proximity   | Type           |
-------------------------------------------------------------
==========================================
|        EQUIPMENT DOWN            |
------------------------------------

**********************************************
 Event Number :  2
_____________________________________________
 Description :  passing_waypoint
_____________________________________________
 Time        :   38.000
**********************************************

>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
```

```
 rule   : WaypointArrival-DepthComparison
 level  : Low_assessment
 action : determine_type_of_depth_change
 time   :   38.268


>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : Waypoint_DistanceTime_Check
 level  : Low_assessment
 action : determine_if_need_to_increase_speed
 time   :   38.444


>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : Waypoint_monitor
 level  : Low_assessment
 action : assess_next_waypoint_and_sequence
 time   :   38.607

============================================================
|              Skipper's Display                           |
============================================================
  TIME  in min_secs     0:38
  Overall Mission Status  >>>> Continue_Unrestricted <<<<
  Manuevering_Status :  unrestricted
  Equipment_Status   :  normal
  Navigation_Status  :  within_tolerance
  Environment_status :  normal
  Spec_Mission_status:  feasible
------------------------------------------------------------|
| evolution     : transit
| depth-status  : dive
| ==========================================================
| Last Command to Guidance :  mark_on_top
| enroute-waypoint         :  3
------------------------------------------------------------
|                Obstacles                                 |
============================================================
|  Direction   |    Proximity   | Type                     |
------------------------------------------------------------
============================================
```

| EQUIPMENT DOWN |

****

Event Number : 3

Description : passing_waypoint

Time : 107.000

****

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Navigation
rule : WaypointArrival-DepthComparison
level : Low_assessment
action : determine_type_of_depth_change
time : 107.276

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Navigation
rule : Waypoint_DistanceTime_Check
level : Low_assessment
action : determine_if_need_to_increase_speed
time : 107.456

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Navigation
rule : Navigation_Assessment
level : Assessment
action : determine_Nav_Status_and_pass_to_Overall_Mission_assessor
time : 107.620

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Navigation
rule : Waypoint_monitor
level : Low_assessment
action : assess_next_waypoint_and_sequence
time : 107.781

==========

```
|              Skipper's Display              |
=============================================================
 TIME in min_secs    1:47
 Overall Mission Status  >>>> Continue_Unrestricted <<<<
 Manuevering_Status :  unrestricted
 Equipment_Status   :  normal
 Navigation_Status  :  within_tolerance
 Environment_status :  normal
 Spec_Mission_status:  feasible
-------------------------------------------------------|
| evolution     : transit
| depth-status  :  no-depth-change
| ======================================================
| Last Command to Guidance :  mark_on_top
| enroute-waypoint        :  4
---------------------------------------------------
|              Obstacles                  |
=============================================================
|  Direction   |    Proximity  | Type           |
-------------------------------------------------
=========================================
|       EQUIPMENT DOWN          |
-----------------------------------

*********************************************
 Event Number :  4
_____________________________________________
 Description :  passing_waypoint
_____________________________________________
 Time       :    125.000
*********************************************

>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   :  Navigation
 rule   :  WaypointArrival-DepthComparison
 level  :  Low_assessment
 action :  determine_type_of_depth_change
 time   :    125.232


>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   :  Navigation
 rule   :  Waypoint_DistanceTime_Check
```

level : Low_assessment
action : determine_if_need_to_increase_speed
time : 125.415

```
============================================================
|              Skipper's Display                 |
============================================================
 TIME in min_secs    2:05
 Overall Mission Status  >>>> Continue_Unrestricted <<<<
 Manuevering_Status :  unrestricted
 Equipment_Status   :  normal
 Navigation_Status  :  within_tolerance
 Environment_status :  normal
 Spec_Mission_status:  feasible
------------------------------------------------------------|
| evolution     : transit
| depth-status  : no-depth-change
| ==========================================================
| Last Command to Guidance :  Increase-Speed
| enroute-waypoint        :  4
-------------------------------------------------------------
|                Obstacles                       |
============================================================
|  Direction   |    Proximity   | Type           |
-------------------------------------------------------------
=============================================
|        EQUIPMENT DOWN             |
---------------------------------------
```

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Navigation
rule : Waypoint_monitor
level : Low_assessment
action : assess_next_waypoint_and_sequence
time : 126.153

****
Event Number : 5

Description : passing_waypoint

Time : 145.000

```
*********************************************

>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : WaypointArrival-DepthComparison
 level  : Low_assessment
 action : determine_type_of_depth_change
 time   :   145.203


>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : Waypoint_DistanceTime_Check
 level  : Low_assessment
 action : determine_if_need_to_increase_speed
 time   :   145.388


>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : Waypoint_monitor
 level  : Low_assessment
 action : assess_next_waypoint_and_sequence
 time   :   145.551

============================================================
|              Skipper's Display              |
============================================================
  TIME  in min_secs     2:25
  Overall Mission Status  >>>> Continue_Unrestricted <<<<
  Manuevering_Status :  unrestricted
  Equipment_Status   :  normal
  Navigation_Status  :  within_tolerance
  Environment_status :  normal
  Spec_Mission_status:  feasible
------------------------------------------------------------|
| evolution     :  specmiss
| depth-status  :  no-depth-change
| ==========================================================
| Last Command to Guidance :  mark_on_top
| enroute-waypoint         :  6
-------------------------------------------------------
|              Obstacles                  |
```

```
============================================================
| Direction  |   Proximity  | Type            |
------------------------------------------------------------
==========================================
|       EQUIPMENT DOWN          |
------------------------------------------

**********************************************
Event Number :  6
_____________________________________________
Description :  passing_waypoint
_____________________________________________
Time       :   167.000
**********************************************

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type   : Navigation
rule   : WaypointArrival-DepthComparison
level  : Low_assessment
action :  determine_type_of_depth_change
time   :   167.201


>>>>>>>>>>>> Decision <<<<<<<<<<<<
type   : Navigation
rule   : Waypoint_DistanceTime_Check
level  : Low_assessment
action :  determine_if_need_to_increase_speed
time   :   167.391


>>>>>>>>>>>> Decision <<<<<<<<<<<<
type   : Navigation
rule   : Waypoint_monitor
level  : Low_assessment
action :  assess_next_waypoint_and_sequence
time   :   167.554

============================================================
|              Skipper's Display              |
============================================================
 TIME  in min_secs     2:47
 Overall Mission Status  >>>> Continue_Unrestricted <<<<
```

```
 Manuevering_Status :  unrestricted
 Equipment_Status   :  normal
 Navigation_Status  :  within_tolerance
 Environment_status :  normal
 Spec_Mission_status:  feasible
-----------------------------------------------------|
| evolution     :  specmiss
| depth-status  :  ascent
| =====================================================
| Last Command to Guidance :  mark_on_top
| enroute-waypoint         :  7
-------------------------------------------------------
|                Obstacles                  |
============================================================
|  Direction   |    Proximity   | Type            |
------------------------------------------------------------
============================================
|        EQUIPMENT DOWN            |
--------------------------------------

**********************************************
 Event Number :  7
_____________________________________________
 Description :  passing_waypoint
_____________________________________________
 Time       :   175.000
**********************************************

>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   :  Navigation
 rule   :  WaypointArrival-DepthComparison
 level  :  Low_assessment
 action :  determine_type_of_depth_change
 time   :   175.227


>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   :  Navigation
 rule   :  Waypoint_DistanceTime_Check
 level  :  Low_assessment
 action :  determine_if_need_to_increase_speed
 time   :   175.418
```

```
>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : Waypoint_monitor
 level  : Low_assessment
 action : assess_next_waypoint_and_sequence
 time   :   175.583

============================================================
|              Skipper's Display                 |
============================================================
  TIME  in min_secs    2:55
  Overall Mission Status  >>>> Continue_Unrestricted <<<<
  Manuevering_Status :  unrestricted
  Equipment_Status   :  normal
  Navigation_Status  :  within_tolerance
  Environment_status :  normal
  Spec_Mission_status:  feasible
------------------------------------------------------------|
| evolution     : transit
| depth-status  : ascent
| ==========================================================
| Last Command to Guidance :  mark_on_top
| enroute-waypoint         :  8
-------------------------------------------------------
|                Obstacles                       |
============================================================
|  Direction   |    Proximity   | Type              |
-------------------------------------------------------
==========================================
|        EQUIPMENT DOWN             |
-------------------------------------

**********************************************
 Event Number :  8
_____________________________________________
 Description :  passing_waypoint
_____________________________________________
 Time       :   196.000
**********************************************

>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
```

```
 rule   : WaypointArrival-DepthComparison
 level  : Low_assessment
 action : determine_type_of_depth_change
 time   :   196.219


>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : Waypoint_DistanceTime_Check
 level  : Low_assessment
 action : determine_if_need_to_increase_speed
 time   :   196.415


>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : Waypoint_monitor
 level  : Low_assessment
 action : assess_next_waypoint_and_sequence
 time   :   196.580

============================================================
|              Skipper's Display                |
============================================================
  TIME  in min_secs     3:16
  Overall Mission Status  >>>> Continue_Unrestricted <<<<
  Manuevering_Status :  unrestricted
  Equipment_Status   :  normal
  Navigation_Status  :  within_tolerance
  Environment_status :  normal
  Spec_Mission_status:  feasible
------------------------------------------------------------|
| evolution     : transit
| depth-status  :  surface
| ==========================================================
| Last Command to Guidance :  mark_on_top
| enroute-waypoint         :  9
-------------------------------------------------------------
|                Obstacles                      |
============================================================
|  Direction   |    Proximity   | Type             |
-------------------------------------------------------------
=============================================
```

```
|       EQUIPMENT DOWN          |
---------------------------------------

*********************************************
Event Number :  9
_____________________________________________
Description :  passing_waypoint
_____________________________________________
Time       :   210.000
*********************************************


>>>>>Made it to Goal<<<<<<<
  At time : 210.03600000000001
18365 rules fired        Run time is 212.2829999999994 seconds
86.51187330120663 rules per second
16 mean number of facts (20 maximum)
2 mean number of activations (5 maximum)
CLIPS> (dribble-off)
```

2. SCENARIO 2

```
CLIPS> (watch statistics)
CLIPS> (batch upload.bat)
CLIPS> (close)
FALSE
CLIPS> (clear)
CLIPS> (load skipper.clp)
CLIPS> (load maneuvering.clp)
CLIPS> (load navigation.clp)
CLIPS> (load sensor.clp)
CLIPS> (load environment.clp)
CLIPS> (reset)
CLIPS> (run)
```

Welcome to the MISSION EXECUTOR DEMO

WAYPOINTS: All scenarios take place over the same set of INITIAL waypoint coordinates.

EQUIPMENT: All equipment is simulated in the event file Objects are created for each onboard equipment

SITUATIONS: All situations are also simulated in the event file. For instance, an obstacle detection is listed and this simulates the Obstacle Avoidance DecisionMaker passing this information through the interface to the Executor .

SCENARIO CHOICES: select number <Ret>

1 Waypoint_Hopping Only (transit)
2 Obstacle Avoidance
3 Vehicle Control System Failure
4 Obstacles and Environment Problems
5 Equipment Failures
6 Exit the Simulator

2

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Navigation
rule : WaypointArrival-DepthComparison
level : Low_assessment
action : determine_type_of_depth_change
time : 0.247

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Navigation
rule : Waypoint_DistanceTime_Check
level : Low_assessment
action : determine_if_need_to_increase_speed
time : 0.423

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Navigation
rule : Waypoint_monitor
level : Low_assessment
action : assess_next_waypoint_and_sequence
time : 0.586

```
============================================================
|              Skipper's Display                |
============================================================
 TIME in min_secs    0:00
 Overall Mission Status  >>>> Continue_Unrestricted <<<<
 Manuevering_Status : unrestricted
 Equipment_Status  : normal
 Navigation_Status : within_tolerance
 Environment_status : normal
 Spec_Mission_status: feasible
------------------------------------------------------------|
| evolution    : transit
| depth-status : dive
| ==========================================================
| Last Command to Guidance : underway
| enroute-waypoint       : 1
------------------------------------------------------------
```

```
|              Obstacles                |
=========================================================
| Direction  |   Proximity  | Type          |
---------------------------------------------------------
=======================================
|      EQUIPMENT DOWN          |
---------------------------------------

***********************************************
Event Number :  1
_____________________________________________
Description :  passing_waypoint_1
_____________________________________________
Time       :   10.000
***********************************************

>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : WaypointArrival-DepthComparison
 level  : Low_assessment
 action :  determine_type_of_depth_change
 time   :   10.193


>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   :  Waypoint_DistanceTime_Check
 level  : Low_assessment
 action :  determine_if_need_to_increase_speed
 time   :   10.368


>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   :  Waypoint_monitor
 level  : Low_assessment
 action :  assess_next_waypoint_and_sequence
 time   :   10.531
```

```
=========================================================
|              Skipper's Display                |
=========================================================
 TIME  in min_secs    0:10
 Overall Mission Status  >>>> Continue_Unrestricted <<<<
 Manuevering_Status :  unrestricted
 Equipment_Status   :  normal
 Navigation_Status  :  within_tolerance
 Environment_status :  normal
 Spec_Mission_status:  feasible
---------------------------------------------------------|
| evolution    : transit
| depth-status : dive
| ======================================================
| Last Command to Guidance :  mark_on_top
| enroute-waypoint        :  2
-----------------------------------------------------------
|               Obstacles                    |
=============================================================
|  Direction  |    Proximity  | Type            |
-------------------------------------------------------------
=========================================
|       EQUIPMENT DOWN           |
-----------------------------------------

*********************************************
 Event Number :  2
_____________________________________________
 Description :  passing_waypoint_2
_____________________________________________
 Time       :   20.000
*********************************************

>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   :  Navigation
 rule   :  WaypointArrival-DepthComparison
 level  :  Low_assessment
 action :  determine_type_of_depth_change
 time   :   20.627
```

```
>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : Waypoint_DistanceTime_Check
 level  : Low_assessment
 action : determine_if_need_to_increase_speed
 time   :   20.804


>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : Waypoint_monitor
 level  : Low_assessment
 action : assess_next_waypoint_and_sequence
 time   :   20.968

============================================================
|              Skipper's Display              |
============================================================
 TIME  in min_secs     0:20
 Overall Mission Status  >>>> Continue_Unrestricted <<<<
 Manuevering_Status :  unrestricted
 Equipment_Status   :  normal
 Navigation_Status  :  within_tolerance
 Environment_status :  normal
 Spec_Mission_status:  feasible
------------------------------------------------------|
| evolution     : transit
| depth-status  : dive
| ====================================================
| Last Command to Guidance :  mark_on_top
| enroute-waypoint         :  3
-------------------------------------------------------
|                Obstacles                  |
============================================================
|  Direction   |    Proximity   | Type             |
------------------------------------------------------------
============================================
|        EQUIPMENT DOWN            |
--------------------------------------
```

```
**********************************************
Event Number : 3
_____________________________________________
Description : obstacles_close_aboard
_____________________________________________
Time      :  27.000
**********************************************

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type  : Maneuvering
rule  : emergency-evasive-maneuver-ascend
level : Low-supervisory-level
action : ascend_to_avoid_obstacle
time  :  27.232

============================================================
|            Skipper's Display              |
============================================================
 TIME in min_secs    0:27
 Overall Mission Status  >>>> Continue_Unrestricted <<<<
 Manuevering_Status : unrestricted
 Equipment_Status   : normal
 Navigation_Status  : within_tolerance
 Environment_status : normal
 Spec_Mission_status: feasible
------------------------------------------------------|
| evolution    : transit
| depth-status : dive
| ====================================================
| Last Command to Guidance : ascend-?*safe_depth*
| enroute-waypoint       : 3
-------------------------------------------------------
|              Obstacles                  |
============================================================
|  Direction  |   Proximity  | Type          |
-------------------------------------------------------
==========================================
|       EQUIPMENT DOWN          |


-----------------------------------
```

```
>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type  : Navigation
 rule  : depth_sounding_deviation_shortrng
 level : low_supervisory
 action : avoid_possible_shoaling
 time  :  27.968

============================================================
|              Skipper's Display                 |
============================================================
 TIME in min_secs    0:27
 Overall Mission Status >>>> Continue_Unrestricted <<<<
 Manuevering_Status : unrestricted
 Equipment_Status  : normal
 Navigation_Status : within_tolerance
 Environment_status : normal
 Spec_Mission_status: feasible
------------------------------------------------------------|
| evolution    : transit
| depth-status : dive
| ==========================================================
| Last Command to Guidance : ascend-?*safe_depth*
| enroute-waypoint       : 3
-------------------------------------------------------------
|                Obstacles                       |
============================================================
|  Direction  |   Proximity  | Type             |
-------------------------------------------------------------
==========================================
|       EQUIPMENT DOWN           |
------------------------------------------

>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type  : Maneuvering
 rule  : emergency_maneuver_evaluation
 level : assessment
 action : assess_emergency_obstacle_avoidance_maneuvers
 time  :  28.704
```

```
**********************************************
Event Number : 4
_____________________________________________
Description : passing_waypoint_3
_____________________________________________
Time      :  35.000
**********************************************

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type  : Navigation
rule  : WaypointArrival-DepthComparison
level : Low_assessment
action : determine_type_of_depth_change
time  :  35.201


>>>>>>>>>>>> Decision <<<<<<<<<<<<
type  : Navigation
rule  : Waypoint_DistanceTime_Check
level : Low_assessment
action : determine_if_need_to_increase_speed
time  :  35.380

============================================================
|              Skipper's Display              |
============================================================
 TIME in min_secs    0:35
 Overall Mission Status  >>>> Continue_Unrestricted <<<<
 Manuevering_Status : unrestricted
 Equipment_Status   : normal
 Navigation_Status  : within_tolerance
 Environment_status : normal
 Spec_Mission_status: feasible
------------------------------------------------------------|
| evolution     : transit
| depth-status  : dive
| ==========================================================
| Last Command to Guidance :  Increase-Speed
| enroute-waypoint        : 3
-------------------------------------------------------
|              Obstacles                    |
============================================================
|  Direction  |   Proximity  | Type          |
```

```
-----------------------------------------------------
=========================================
|       EQUIPMENT DOWN          |
-----------------------------------------

>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type  : Navigation
 rule  : Waypoint_monitor
 level : Low_assessment
 action : assess_next_waypoint_and_sequence
 time  :   36.133


**********************************************
 Event Number : 5
_____________________________________________
 Description : obstacle_detected_at_normal_range
_____________________________________________
 Time      :   41.000
**********************************************

**********************************************
 Event Number : 6
_____________________________________________
 Description : obstacle_classified_as_new
_____________________________________________
 Time      :   41.000
**********************************************

>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type  : Maneuvering
 rule  : Obstacle_detection_Normal_Limits
 level : Low_assessment
 action : classify_normal_range_obstacle_as_new
 time  :   41.439


>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type  : Maneuvering
 rule  : Collective_Obstacle_Assessment
 level : Low_assessment
 action : assess_whether_presents_a_collision_danger_and_turn
 time  :   41.650
```

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Maneuvering
rule : Maneuvering_Status_Assessment
level : maneuvering-assessment
action : change-overall-maneuvering-status
time : 41.817

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Overall_Mission
rule : Overall_Mission_Assessor
level : High
action : ContinueMission_with_restrictions
time : 41.988

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Overall_Mission
rule : Continue_mission_restricted_initial
level : Assessment
action : Note-time-of-status-change
time : 42.149

****
Event Number : 7

Description : obstacle_detected_at_normal_range

Time : 41.000
****

****
Event Number : 8

Description : obstacle_classified_as_new

Time : 41.000
****

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Maneuvering
rule : Obstacle_detection_Normal_Limits
level : Low_assessment
action : classify_normal_range_obstacle_as_new
time : 42.786

****
Event Number : 9

Description : obstacle_detected_at_normal_range

Time : 45.000
****

****
Event Number : 10

Description : obstacle_classified_as_new

Time : 45.000
****

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Maneuvering
rule : Obstacle_detection_Normal_Limits
level : Low_assessment
action : classify_normal_range_obstacle_as_new
time : 45.453

****
Event Number : 11

Description : obstacle_detected_at_normal_range

Time : 50.000
****

****
Event Number : 12

Description : obstacle_classified_as_new

Time : 50.000

****

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Maneuvering
rule : Obstacle_detection_Normal_Limits
level : Low_assessment
action : classify_normal_range_obstacle_as_new
time : 50.452

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Maneuvering
rule : Maneuvering_Status_Assessment
level : maneuvering-assessment
action : change-overall-maneuvering-status
time : 57.079

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Overall_Mission
rule : Overall_Mission_Assessor
level : High
action : Abort_mission
time : 57.251

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Overall_Mission
rule : Abort_Mission
level : Low
action : lock_status_and_replan_route_to_abort_rendezvous
time : 57.407

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Navigation
rule : WaypointArrival-DepthComparison
level : Low_assessment
action : determine_type_of_depth_change

time : 57.718

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Navigation
rule : Waypoint_DistanceTime_Check
level : Low_assessment
action : determine_if_need_to_increase_speed
time : 57.891

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Navigation
rule : Waypoint_monitor
level : Low_assessment
action : assess_next_waypoint_and_sequence
time : 58.055

```
==========================================================
|              Skipper's Display                |
==========================================================
 TIME in min_secs    0:57
 Overall Mission Status  >>>> Abort_mission <<<<
 Manuevering_Status :  severely_restricted
 Equipment_Status   :  normal
 Navigation_Status  :  within_tolerance
 Environment_status :  normal
 Spec_Mission_status:  feasible
----------------------------------------------------------|
| evolution     : abort_transit
| depth-status  : no-depth-change
| ========================================================
| Last Command to Guidance :  mark_on_top
| enroute-waypoint        :  1
----------------------------------------------------------
|                Obstacles                  |
==========================================================
|  Direction   |    Proximity  | Type          |
----------------------------------------------------------
      89.0          far              floating

      356.0         far              floating
```

```
=======================================
|       EQUIPMENT DOWN          |
---------------------------------------

*********************************************
Event Number :  13
_____________________________________________
Description :  mark_on_abort_waypoint
_____________________________________________
Time       :   77.000
*********************************************

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type   : Navigation
rule   : WaypointArrival-DepthComparison
level  : Low_assessment
action :  determine_type_of_depth_change
time   :   77.199


>>>>>>>>>>>> Decision <<<<<<<<<<<<
type   : Navigation
rule   : Waypoint_DistanceTime_Check
level  : Low_assessment
action :  determine_if_need_to_increase_speed
time   :   77.375


>>>>>>>>>>>> Decision <<<<<<<<<<<<
type   : Navigation
rule   : Waypoint_monitor
level  : Low_assessment
action :  assess_next_waypoint_and_sequence
time   :   77.539

=============================================================
|              Skipper's Display              |
=============================================================
 TIME  in min_secs     1:17
 Overall Mission Status  >>>> Abort_mission <<<<
 Manuevering_Status :  severely_restricted
 Equipment_Status   :  normal
 Navigation_Status  :  within_tolerance
```

```
 Environment_status :  normal
 Spec_Mission_status:  feasible
-----------------------------------------------------|
| evolution     : abort_transit
| depth-status  : ascent
| =====================================================
| Last Command to Guidance :  transiting_to_abort_rendezvous
| enroute-waypoint        : 2
-------------------------------------------------
|              Obstacles              |
=======================================================
|  Direction  |   Proximity  | Type          |
-------------------------------------------------------
      89.0          far             floating

      356.0         far             floating

=========================================
|      EQUIPMENT DOWN           |
-----------------------------------------

*********************************************
 Event Number :  14
_____________________________________________
 Description :  mark_on_abort_waypoint
_____________________________________________
 Time       :   98.000
*********************************************

>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : WaypointArrival-DepthComparison
 level  : Low_assessment
 action :  determine_type_of_depth_change
 time   :   98.230


>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : Waypoint_DistanceTime_Check
 level  : Low_assessment
 action :  determine_if_need_to_increase_speed
 time   :   98.406
```

```
>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : Waypoint_monitor
 level  : Low_assessment
 action : assess_next_waypoint_and_sequence
 time   :   98.570

============================================================
|              Skipper's Display              |
============================================================
  TIME  in min_secs     1:38
  Overall Mission Status  >>>> Abort_mission <<<<
  Manuevering_Status :  severely_restricted
  Equipment_Status   :  normal
  Navigation_Status  :  within_tolerance
  Environment_status :  normal
  Spec_Mission_status:  feasible
------------------------------------------------------------|
| evolution     : abort_transit
| depth-status  : ascent
| ==========================================================
| Last Command to Guidance :  underway
| enroute-waypoint         :  3
-------------------------------------------------------------
|                Obstacles                    |
============================================================
|  Direction   |    Proximity   | Type            |
-------------------------------------------------------------
      89.0          far              floating

      356.0         far              floating

========================================
|        EQUIPMENT DOWN          |
----------------------------------------
```

```
**********************************************
Event Number :  15
_____________________________________________
Description :  mark_on_abort_waypoint
_____________________________________________
Time       :   112.000
**********************************************

>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : WaypointArrival-DepthComparison
 level  : Low_assessment
 action :  determine_type_of_depth_change
 time   :   112.246


>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : Waypoint_DistanceTime_Check
 level  : Low_assessment
 action :  determine_if_need_to_increase_speed
 time   :   112.425


>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : Waypoint_monitor
 level  : Low_assessment
 action :  assess_next_waypoint_and_sequence
 time   :   112.589

============================================================
|              Skipper's Display              |
============================================================
  TIME  in min_secs     1:52
  Overall Mission Status  >>>> Abort_mission <<<<
  Manuevering_Status :  severely_restricted
  Equipment_Status   :  normal
  Navigation_Status  :  within_tolerance
  Environment_status :  normal
  Spec_Mission_status:  feasible
------------------------------------------------------------|
```

```
| evolution     : abort_transit
| depth-status  : surface
| ========================================================
| Last Command to Guidance :  mark_on_top
| enroute-waypoint        : 4
---------------------------------------------------------
|                Obstacles                    |
==========================================================
|  Direction   |    Proximity  | Type            |
---------------------------------------------------------
      89.0          far              floating

      356.0         far              floating

=========================================
|        EQUIPMENT DOWN            |
-----------------------------------------

***********************************************
 Event Number :  16
_____________________________________________
 Description :  mark_on_abort_waypoint
_____________________________________________
 Time       :   125.000
***********************************************


>>>>>Made it to Goal<<<<<<<
  At time : 125.0339999999997
10135 rules fired       Run time is 128.4349999999995 seconds
78.91151165959467 rules per second
22 mean number of facts (30 maximum)
2 mean number of activations (8 maximum)
CLIPS> (exit)
```

3. SCENARIO 3

```
CLIPS> (watch statistics)
CLIPS> (batch upload.bat)
CLIPS> (close)
FALSE
CLIPS> (clear)
CLIPS> (load skipper.clp)
CLIPS> (load maneuvering.clp)
CLIPS> (load navigation.clp)
CLIPS> (load sensor.clp)
CLIPS> (load environment.clp)
CLIPS> (reset)
CLIPS> (run)
```

Welcome to the MISSION EXECUTOR DEMO

WAYPOINTS: All scenarios take place over the same set of INITIAL waypoint coordinates.

EQUIPMENT: All equipment is simulated in the event file Objects are created for each onboard equipment

SITUATIONS: All situations are also simulated in the event file. For instance, an obstacle detection is listed and this simulates the Obstacle Avoidance DecisionMaker passing this information through the interface to the Executor .

SCENARIO CHOICES: select number <Ret>

1 Waypoint_Hopping Only (transit)
2 Obstacle Avoidance
3 Vehicle Control System Failure
4 Obstacles and Environment Problems
5 Equipment Failures
6 Exit the Simulator

3

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Navigation
rule : WaypointArrival-DepthComparison
level : Low_assessment
action : determine_type_of_depth_change
time : 0.203

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Navigation
rule : Waypoint_DistanceTime_Check
level : Low_assessment
action : determine_if_need_to_increase_speed
time : 0.380

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Navigation
rule : Waypoint_monitor
level : Low_assessment
action : assess_next_waypoint_and_sequence
time : 0.543

```
============================================================
|              Skipper's Display              |
============================================================
 TIME in min_secs    0:00
 Overall Mission Status >>>> Continue_Unrestricted <<<<
 Manuevering_Status : unrestricted
 Equipment_Status  : normal
 Navigation_Status : within_tolerance
 Environment_status : normal
 Spec_Mission_status: feasible
------------------------------------------------------------|
| evolution    : transit
| depth-status : dive
```

```
| ========================================================
| Last Command to Guidance :  underway
| enroute-waypoint        :  1
---------------------------------------------------------
|                Obstacles                    |
==========================================================
|  Direction   |    Proximity   | Type              |
---------------------------------------------------------
=======================================
|        EQUIPMENT DOWN           |
---------------------------------------

***********************************************
 Event Number :  1
_____________________________________________
 Description :  passing_waypoint_1
_____________________________________________
 Time       :   10.000
***********************************************

>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : WaypointArrival-DepthComparison
 level  : Low_assessment
 action :  determine_type_of_depth_change
 time   :   10.246


>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : Waypoint_DistanceTime_Check
 level  : Low_assessment
 action :  determine_if_need_to_increase_speed
 time   :   10.420


>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : Waypoint_monitor
 level  : Low_assessment
 action :  assess_next_waypoint_and_sequence
 time   :   10.583
```

```
==========================================================
|              Skipper's Display                |
==========================================================
 TIME  in min_secs    0:10
 Overall Mission Status  >>>> Continue_Unrestricted <<<<
 Manuevering_Status :  unrestricted
 Equipment_Status   :  normal
 Navigation_Status  :  within_tolerance
 Environment_status :  normal
 Spec_Mission_status:  feasible
----------------------------------------------------------|
| evolution     :  transit
| depth-status  :  dive
| ========================================================
| Last Command to Guidance :  mark_on_top
| enroute-waypoint         :  2
-----------------------------------------------------------
|                Obstacles                    |
==========================================================
|  Direction   |    Proximity   | Type              |
-----------------------------------------------------------
==========================================
|        EQUIPMENT DOWN            |
------------------------------------

**********************************************
 Event Number :  2
_____________________________________________
 Description :  passing_waypoint_2
_____________________________________________
 Time       :   20.000
**********************************************

>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   :  Navigation
 rule   :  WaypointArrival-DepthComparison
 level  :  Low_assessment
 action :  determine_type_of_depth_change
 time   :   20.616
```

```
>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : Waypoint_DistanceTime_Check
 level  : Low_assessment
 action : determine_if_need_to_increase_speed
 time   :   20.793


>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : Waypoint_monitor
 level  : Low_assessment
 action : assess_next_waypoint_and_sequence
 time   :   20.957

============================================================
|              Skipper's Display                 |
============================================================
  TIME  in min_secs     0:20
  Overall Mission Status  >>>> Continue_Unrestricted <<<<
  Manuevering_Status :  unrestricted
  Equipment_Status   :  normal
  Navigation_Status  :  within_tolerance
  Environment_status :  normal
  Spec_Mission_status:  feasible
------------------------------------------------------------|
| evolution     : transit
| depth-status  : dive
| ==========================================================
| Last Command to Guidance :  mark_on_top
| enroute-waypoint         :  3
------------------------------------------------------------
|              Obstacles                     |
============================================================
|  Direction   |    Proximity   | Type             |
------------------------------------------------------------
==========================================
|        EQUIPMENT DOWN            |
------------------------------------
```

```
**********************************************
Event Number : 3
_____________________________________________
Description : passing_waypoint_3
_____________________________________________
Time      :   27.000
**********************************************

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type  : Navigation
rule  : WaypointArrival-DepthComparison
level : Low_assessment
action : determine_type_of_depth_change
time  :  27.201


>>>>>>>>>>>> Decision <<<<<<<<<<<<
type  : Navigation
rule  : Waypoint_DistanceTime_Check
level : Low_assessment
action : determine_if_need_to_increase_speed
time  :  27.383

============================================================
|              Skipper's Display              |
============================================================
 TIME  in min_secs    0:27
 Overall Mission Status  >>>> Continue_Unrestricted <<<<
 Manuevering_Status :  unrestricted
 Equipment_Status   :  normal
 Navigation_Status  :  within_tolerance
 Environment_status :  normal
 Spec_Mission_status:  feasible
-----------------------------------------------------|
| evolution    : transit
| depth-status : dive
| ===================================================
| Last Command to Guidance :  Increase-Speed
| enroute-waypoint       : 3
-----------------------------------------------------
|              Obstacles                  |
============================================================
```

```
| Direction  |   Proximity  | Type          |
-----------------------------------------------------
=======================================
|       EQUIPMENT DOWN          |
---------------------------------------

>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : Waypoint_monitor
 level  : Low_assessment
 action : assess_next_waypoint_and_sequence
 time   :   28.122


**********************************************
 Event Number :  4
_____________________________________________
 Description :  Plane_controls_failure
_____________________________________________
 Time       :   35.000
**********************************************

>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Maneuvering
 rule   : Maneuvering_Status_Assessment
 level  : maneuvering-assessment
 action : change-overall-maneuvering-status
 time   :   35.243


>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : System_Monitor
 rule   : Control_System_Failure
 level  : Low
 action : Pass_info_to_Equip_Assessor
 time   :   35.409


>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : System_Monitor
 rule   : Equipment_Status_Assessment
 level  : Assessment
 action : Assessing_Status
```

time : 35.571

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Overall_Mission
rule : Overall_Mission_Assessor
level : High
action : Abort_mission
time : 35.740

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Overall_Mission
rule : Abort_Mission
level : Low
action : lock_status_and_replan_route_to_abort_rendezvous
time : 35.898

>>>>> Shutting Down for Dynamic Recovery <<<
>>>>> Transponder will function for 2 hours <<<
2931 rules fired Run time is 41.71299999999974 seconds
70.26586435883343 rules per second
16 mean number of facts (27 maximum)
2 mean number of activations (11 maximum)
CLIPS> (dribble-off)

4. SCENARIO 4

```
CLIPS> (watch statistics)
CLIPS> (batch upload.bat)
CLIPS> (close)
FALSE
CLIPS> (clear)
CLIPS> (load skipper.clp)
CLIPS> (load maneuvering.clp)
CLIPS> (load navigation.clp)
CLIPS> (load sensor.clp)
CLIPS> (load environment.clp)
CLIPS> (reset)
CLIPS> (run)
```

Welcome to the MISSION EXECUTOR DEMO

WAYPOINTS: All scenarios take place over the same set of INITIAL waypoint coordinates.

EQUIPMENT: All equipment is simulated in the event file Objects are created for each onboard equipment

SITUATIONS: All situations are also simulated in the event file. For instance, an obstacle detection is listed and this simulates the Obstacle Avoidance DecisionMaker passing this information through the interface to the Executor .

SCENARIO CHOICES: select number <Ret>

1 Waypoint_Hopping Only (transit)
2 Obstacle Avoidance
3 Vehicle Control System Failure
4 Obstacles and Environment Problems
5 Equipment Failures
6 Exit the Simulator

4

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Navigation
rule : WaypointArrival-DepthComparison
level : Low_assessment
action : determine_type_of_depth_change
time : 0.217

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Navigation
rule : Waypoint_DistanceTime_Check
level : Low_assessment
action : determine_if_need_to_increase_speed
time : 0.392

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Navigation
rule : Waypoint_monitor
level : Low_assessment
action : assess_next_waypoint_and_sequence
time : 0.557

```
============================================================
|              Skipper's Display              |
============================================================
 TIME in min_secs    0:00
 Overall Mission Status >>>> Continue_Unrestricted <<<<
 Manuevering_Status : unrestricted
 Equipment_Status  : normal
 Navigation_Status : within_tolerance
 Environment_status : normal
 Spec_Mission_status: feasible
------------------------------------------------------------|
| evolution    : transit
| depth-status : dive
| ==========================================================
| Last Command to Guidance : underway
| enroute-waypoint       : 1
```

```
-----------------------------------------------------
|              Obstacles                            |
=====================================================
| Direction  |  Proximity  | Type          |
-----------------------------------------------------
=========================================
|      EQUIPMENT DOWN          |
-----------------------------------

***********************************************
Event Number :  1
_____________________________________________
Description :  passing_waypoint
_____________________________________________
Time      :   10.000
***********************************************

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type   :  Navigation
rule   :  WaypointArrival-DepthComparison
level  :  Low_assessment
action :  determine_type_of_depth_change
time   :   10.214


>>>>>>>>>>>> Decision <<<<<<<<<<<<
type   :  Navigation
rule   :  Waypoint_DistanceTime_Check
level  :  Low_assessment
action :  determine_if_need_to_increase_speed
time   :   10.388


>>>>>>>>>>>> Decision <<<<<<<<<<<<
type   :  Navigation
rule   :  Waypoint_monitor
level  :  Low_assessment
action :  assess_next_waypoint_and_sequence
time   :   10.552

=========================================================
|              Skipper's Display                        |
=========================================================
```

```
 TIME  in min_secs    0:10
 Overall Mission Status  >>>> Continue_Unrestricted <<<<
 Manuevering_Status :  unrestricted
 Equipment_Status   :  normal
 Navigation_Status  :  within_tolerance
 Environment_status :  normal
 Spec_Mission_status:  feasible
-----------------------------------------------------------|
| evolution     : transit
| depth-status  : dive
| ==========================================================
| Last Command to Guidance :  mark_on_top
| enroute-waypoint        :  2
-------------------------------------------------------
|                Obstacles                    |
============================================================
|  Direction   |    Proximity   | Type             |
-------------------------------------------------------
==========================================
|        EQUIPMENT DOWN            |
------------------------------------

**********************************************
 Event Number :  2
_____________________________________________
 Description :  obstacle_detected_at_normal_range
_____________________________________________
 Time       :   27.000
**********************************************

**********************************************
 Event Number :  3
_____________________________________________
 Description :  obstacle_classified_as_new
_____________________________________________
 Time       :   27.000
**********************************************

>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   :  Maneuvering
 rule   :  Obstacle_detection_Normal_Limits
 level  :  Low_assessment
 action :  classify_normal_range_obstacle_as_new
```

time : 27.428

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Maneuvering
rule : Collective_Obstacle_Assessment
level : Low_assessment
action : assess_whether_presents_a_collision_danger_and_turn
time : 27.624

****
Event Number : 4

Description : obstacle_detected_at_normal_range

Time : 37.000
****

****
Event Number : 5

Description : obstacle_detected_at_normal_range

Time : 37.000
****

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Maneuvering
rule : Obstacle_detection_Normal_Limits
level : Low_assessment
action : classify_normal_range_obstacle_as_new
time : 37.475

****
Event Number : 6

Description : obstacle_classified_as_new

Time : 40.000
****

```
**********************************************
Event Number :  7
_____________________________________________
Description :  obstacle_update_previous_detect
_____________________________________________
Time       :   40.000
**********************************************

>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   :  Maneuvering
 rule   :  Obstacle_Update
 level  :  Low_assessment
 action :  update_obstacle_status:rangebearing,collision-danger
 time   :   40.843


**********************************************
Event Number :  8
_____________________________________________
Description :  sea_temp_does_not_match_exp_reading
_____________________________________________
Time       :   50.000
**********************************************

>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   :  Environmental_world
 rule   :  Environment_Assessor
 level  :  Assessment
 action :  determine_if_environment_status_is_hazard
 time   :   50.256


**********************************************
Event Number :  9
_____________________________________________
Description :  pressure_out_of_limits
_____________________________________________
Time       :   60.000
**********************************************

>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   :  Environmental_world
 rule   :  Environment_Assessor
```

level : Assessment
action : determine_if_environment_status_is_hazard
time : 60.750

****
Event Number : 10

Description : gyro_indicates_abnormal_sea_turbulence

Time : 65.000
****

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Environmental_world
rule : Environment_Assessor
level : Assessment
action : determine_if_environment_status_is_hazard
time : 65.244

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Overall_Mission
rule : Overall_Mission_Assessor
level : High
action : Abort_mission
time : 65.411

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Overall_Mission
rule : Abort_Mission
level : Low
action : lock_status_and_replan_route_to_abort_rendezvous
time : 65.569

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Navigation
rule : WaypointArrival-DepthComparison
level : Low_assessment
action : determine_type_of_depth_change
time : 65.867

```
>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : Waypoint_DistanceTime_Check
 level  : Low_assessment
 action : determine_if_need_to_increase_speed
 time   :   66.040


>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : Waypoint_monitor
 level  : Low_assessment
 action : assess_next_waypoint_and_sequence
 time   :   66.203

============================================================
|              Skipper's Display                 |
============================================================
  TIME  in min_secs     1:05
  Overall Mission Status  >>>> Abort_mission <<<<
  Manuevering_Status :  unrestricted
  Equipment_Status   :  normal
  Navigation_Status  :  within_tolerance
  Environment_status :  major_deviation
  Spec_Mission_status:  feasible
------------------------------------------------------------|
| evolution     : abort_transit
| depth-status  : no-depth-change
| ==========================================================
| Last Command to Guidance :  transiting_to_abort_rendezvous
| enroute-waypoint        :  1
-------------------------------------------------------------
|                Obstacles                       |
============================================================
|  Direction   |    Proximity   | Type           |
-------------------------------------------------------------
       82.0          far              floating

       356.0         far              floating

============================================
```

```
|       EQUIPMENT DOWN          |
---------------------------------------
>>>>>>>> SeaTempSensor<<<<<<<<
>>>>>>>> SeaStateGyro<<<<<<<<
>>>>>>>> PressureTransducer<<<<<<<<

**********************************************
Event Number : 11
_____________________________________________
Description : passing_waypoint
_____________________________________________
Time      :  95.000
**********************************************

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type  : Navigation
rule  : WaypointArrival-DepthComparison
level : Low_assessment
action : determine_type_of_depth_change
time  :  95.225


>>>>>>>>>>>> Decision <<<<<<<<<<<<
type  : Navigation
rule  : Waypoint_DistanceTime_Check
level : Low_assessment
action : determine_if_need_to_increase_speed
time  :  95.399


>>>>>>>>>>>> Decision <<<<<<<<<<<<
type  : Navigation
rule  : Waypoint_monitor
level : Low_assessment
action : assess_next_waypoint_and_sequence
time  :  95.565

============================================================
|              Skipper's Display              |
============================================================
 TIME in min_secs    1:35
 Overall Mission Status  >>>> Abort_mission <<<<
 Manuevering_Status : unrestricted
```

```
 Equipment_Status  : normal
 Navigation_Status : within_tolerance
 Environment_status : major_deviation
 Spec_Mission_status: feasible
-----------------------------------------------------|
| evolution    : abort_transit
| depth-status : ascent
| ===================================================
| Last Command to Guidance : underway
| enroute-waypoint       : 2
-------------------------------------------------------
|              Obstacles                  |
=========================================================
|  Direction  |   Proximity  | Type          |
---------------------------------------------------------
      82.0         far            floating

      356.0        far            floating

=========================================
|       EQUIPMENT DOWN          |
-----------------------------------------
>>>>>>>> SeaTempSensor<<<<<<<<
>>>>>>>> SeaStateGyro<<<<<<<<
>>>>>>>> PressureTransducer<<<<<<<<

***********************************************
 Event Number : 12
_____________________________________________
 Description : passing_waypoint
_____________________________________________
 Time      :   115.000
***********************************************

>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type  : Navigation
 rule  : WaypointArrival-DepthComparison
 level : Low_assessment
 action : determine_type_of_depth_change
 time  :   115.240
```

```
>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : Waypoint_DistanceTime_Check
 level  : Low_assessment
 action : determine_if_need_to_increase_speed
 time   :   115.417


>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : Waypoint_monitor
 level  : Low_assessment
 action : assess_next_waypoint_and_sequence
 time   :   115.582

=========================================================
|              Skipper's Display                        |
=========================================================
  TIME  in min_secs     1:55
  Overall Mission Status  >>>> Abort_mission <<<<
  Manuevering_Status :  unrestricted
  Equipment_Status   :  normal
  Navigation_Status  :  within_tolerance
  Environment_status :  major_deviation
  Spec_Mission_status:  feasible
---------------------------------------------------------|
| evolution     :  abort_transit
| depth-status  :  ascent
| ========================================================
| Last Command to Guidance :  mark_on_top
| enroute-waypoint         :  3
-----------------------------------------------------
|               Obstacles                       |
=========================================================
|  Direction   |    Proximity  | Type           |
---------------------------------------------------------
       82.0          far              floating

       356.0         far              floating

=========================================
|        EQUIPMENT DOWN            |
-----------------------------------------
```

>>>>>>>> SeaTempSensor<<<<<<<<
>>>>>>>> SeaStateGyro<<<<<<<<
>>>>>>>> PressureTransducer<<<<<<<<

****
Event Number : 13

Description : passing_waypoint

Time : 126.000
****

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Navigation
rule : WaypointArrival-DepthComparison
level : Low_assessment
action : determine_type_of_depth_change
time : 126.214

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Navigation
rule : Waypoint_DistanceTime_Check
level : Low_assessment
action : determine_if_need_to_increase_speed
time : 126.392

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Navigation
rule : Waypoint_monitor
level : Low_assessment
action : assess_next_waypoint_and_sequence
time : 126.557

==========
| Skipper's Display |
==========
TIME in min_secs 2:06
Overall Mission Status >>>> Abort_mission <<<<
Manuevering_Status : unrestricted
Equipment_Status : normal
Navigation_Status : within_tolerance

```
 Environment_status :  major_deviation
 Spec_Mission_status:  feasible
-----------------------------------------------------|
| evolution    : abort_transit
| depth-status  : surface
| =====================================================
| Last Command to Guidance :  mark_on_top
| enroute-waypoint        :  4
-------------------------------------------------------
|                Obstacles                    |
============================================================
|  Direction   |    Proximity  | Type             |
------------------------------------------------------------
      82.0          far              floating

      356.0         far              floating

=========================================
|        EQUIPMENT DOWN            |
-----------------------------------------
>>>>>>>>  SeaTempSensor<<<<<<<<
>>>>>>>>  SeaStateGyro<<<<<<<<
>>>>>>>>  PressureTransducer<<<<<<<<

*********************************************
 Event Number :  14
_____________________________________________
 Description :  Goal_arrival
_____________________________________________
 Time       :   148.000
*********************************************


>>>>>Made it to Goal<<<<<<<
  At time : 148.0519999999997
13301 rules fired       Run time is 150.860999999999 seconds
88.16725329939541 rules per second
20 mean number of facts (27 maximum)
2 mean number of activations (7 maximum)
CLIPS> (dribble-off)
```

5. SCENARIO 5

```
CLIPS> (watch statistics)
CLIPS> (batch upload.bat)
CLIPS> (close)
FALSE
CLIPS> (clear)
CLIPS> (load skipper.clp)
CLIPS> (load maneuvering.clp)
DCLIPS> (load navigation.clp)
DCLIPS> (load sensor.clp)
CLIPS> (load environment.clp)
j
CLIPS> (reset)
CLIPS> (run)
```

Welcome to the MISSION EXECUTOR DEMO

WAYPOINTS: All scenarios take place over the same set of INITIAL waypoint coordinates.

EQUIPMENT: All equipment is simulated in the event file Objects are created for each onboard equipment

SITUATIONS: All situations are also simulated in the event file. For instance, an obstacle detection is listed and this simulates the Obstacle Avoidance DecisionMaker passing this information through the interface to the Executor .

SCENARIO CHOICES: select number <Ret>
1 Waypoint_Hopping Only (transit)
2 Obstacle Avoidance
3 Vehicle Control System Failure
4 Obstacles and Environment Problems
5 Equipment Failures

6 Exit the Simulator

5

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Navigation
rule : WaypointArrival-DepthComparison
level : Low_assessment
action : determine_type_of_depth_change
time : 0.202

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Navigation
rule : Waypoint_DistanceTime_Check
level : Low_assessment
action : determine_if_need_to_increase_speed
time : 0.376

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Navigation
rule : Waypoint_monitor
level : Low_assessment
action : assess_next_waypoint_and_sequence
time : 0.539

```
============================================================
|              Skipper's Display                 |
============================================================
 TIME in min_secs    0:00
 Overall Mission Status >>>> Continue_Unrestricted <<<<
 Manuevering_Status : unrestricted
 Equipment_Status  : normal
 Navigation_Status : within_tolerance
 Environment_status : normal
 Spec_Mission_status: feasible
------------------------------------------------------------|
| evolution    : transit
```

```
| depth-status : dive
| ========================================================
| Last Command to Guidance : underway
| enroute-waypoint       : 1
---------------------------------------------------------
|              Obstacles                    |
============================================================
|  Direction  |   Proximity  | Type          |
-------------------------------------------------------
==========================================
|       EQUIPMENT DOWN          |
------------------------------------

**********************************************
 Event Number : 1
_____________________________________________
 Description : passing_waypoint_1
_____________________________________________
 Time      :   10.000
**********************************************

>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : WaypointArrival-DepthComparison
 level  : Low_assessment
 action : determine_type_of_depth_change
 time   :   10.216


>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : Waypoint_DistanceTime_Check
 level  : Low_assessment
 action : determine_if_need_to_increase_speed
 time   :   10.390


>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : Waypoint_monitor
 level  : Low_assessment
 action : assess_next_waypoint_and_sequence
 time   :   10.553
```

```
==========================================================
|              Skipper's Display                |
==========================================================
 TIME  in min_secs    0:10
 Overall Mission Status  >>>> Continue_Unrestricted <<<<
 Manuevering_Status :  unrestricted
 Equipment_Status   :  normal
 Navigation_Status  :  within_tolerance
 Environment_status :  normal
 Spec_Mission_status:  feasible
----------------------------------------------------------|
| evolution     : transit
| depth-status  : dive
| ========================================================
| Last Command to Guidance :  mark_on_top
| enroute-waypoint        :  2
-----------------------------------------------------------
|                Obstacles                    |
==========================================================
|  Direction   |    Proximity   | Type             |
-----------------------------------------------------------
==========================================
|        EQUIPMENT DOWN            |
------------------------------------

**********************************************
 Event Number :  2
_____________________________________________
 Description :  passing_waypoint_2
_____________________________________________
 Time       :   20.000
**********************************************

>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   :  Navigation
 rule   :  WaypointArrival-DepthComparison
 level  :  Low_assessment
 action :  determine_type_of_depth_change
 time   :   20.634
```

```
>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : Waypoint_DistanceTime_Check
 level  : Low_assessment
 action : determine_if_need_to_increase_speed
 time   :   20.811


>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : Waypoint_monitor
 level  : Low_assessment
 action : assess_next_waypoint_and_sequence
 time   :   20.975

============================================================
|              Skipper's Display                 |
============================================================
 TIME  in min_secs     0:20
 Overall Mission Status  >>>> Continue_Unrestricted <<<<
 Manuevering_Status :  unrestricted
 Equipment_Status   :  normal
 Navigation_Status  :  within_tolerance
 Environment_status :  normal
 Spec_Mission_status:  feasible
------------------------------------------------------------|
| evolution     : transit
| depth-status  : dive
| ==========================================================
| Last Command to Guidance :  mark_on_top
| enroute-waypoint         :  3
-------------------------------------------------------------
|                Obstacles                       |
============================================================
|  Direction   |    Proximity   | Type              |
-------------------------------------------------------------
=========================================
|        EQUIPMENT DOWN            |
-----------------------------------------
```

```
**********************************************
Event Number : 3
_____________________________________________
Description : passing_waypoint_3
_____________________________________________
Time      :  27.000
**********************************************

>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type  : Navigation
 rule  : WaypointArrival-DepthComparison
 level : Low_assessment
 action : determine_type_of_depth_change
 time  :  27.209


>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type  : Navigation
 rule  : Waypoint_DistanceTime_Check
 level : Low_assessment
 action : determine_if_need_to_increase_speed
 time  :  27.389

============================================================
|              Skipper's Display              |
============================================================
 TIME  in min_secs    0:27
 Overall Mission Status  >>>> Continue_Unrestricted <<<<
 Manuevering_Status :  unrestricted
 Equipment_Status  :  normal
 Navigation_Status  :  within_tolerance
 Environment_status :  normal
 Spec_Mission_status:  feasible
-----------------------------------------------------|
| evolution    : transit
| depth-status  : dive
| ===================================================
| Last Command to Guidance :  Increase-Speed
| enroute-waypoint       : 3
-----------------------------------------------------
|              Obstacles                    |
============================================================
|  Direction  |   Proximity  | Type           |
```

```
-------------------------------------------------
=================================================
|        EQUIPMENT DOWN              |
-------------------------------------

>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : Waypoint_monitor
 level  : Low_assessment
 action : assess_next_waypoint_and_sequence
 time   :  28.129


************************************************
 Event Number :  4
_________________________________________________
 Description :  sonar_has-failure-reading
_________________________________________________
 Time       :  35.000
************************************************

>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : System_Monitor
 rule   : Sonar_Failure
 level  : Low
 action : Pass_info_to_Equip_Assessor
 time   :  35.300


>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : System_Monitor
 rule   : Equipment_Status_Assessment
 level  : Assessment
 action : Assessing_Status
 time   :  35.463


>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Overall_Mission
 rule   : Overall_Mission_Assessor
 level  : High
 action : ContinueMission_with_restrictions
 time   :  35.630
```

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Overall_Mission
rule : Continue_mission_restricted_initial
level : Assessment
action : Note-time-of-status-change
time : 35.791

****
Event Number : 5

Description : passing_waypoint_4

Time : 55.000
****

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Navigation
rule : WaypointArrival-DepthComparison
level : Low_assessment
action : determine_type_of_depth_change
time : 55.236

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Navigation
rule : Waypoint_DistanceTime_Check
level : Low_assessment
action : determine_if_need_to_increase_speed
time : 55.419

==========
| Skipper's Display |
==========
TIME in min_secs 0:55
Overall Mission Status >>>> Continue_with_Restrictions <<<<
Manuevering_Status : unrestricted
Equipment_Status : equipment_critical
Navigation_Status : within_tolerance
Environment_status : normal
Spec_Mission_status: feasible
---|
| evolution : transit

```
| depth-status : no-depth-change
| ========================================================
| Last Command to Guidance : Increase-Speed
| enroute-waypoint       : 4
---------------------------------------------------------
|                Obstacles                    |
============================================================
|  Direction   |   Proximity  | Type          |
---------------------------------------------------------
=========================================
|       EQUIPMENT DOWN          |
-----------------------------------------
>>>>>>>> FWD-sonar<<<<<<<<

>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Navigation
 rule   : Waypoint_monitor
 level  : Low_assessment
 action : assess_next_waypoint_and_sequence
 time   :  56.202


**********************************************
 Event Number : 6
_____________________________________________
 Description : sonar_has_failure-reading
_____________________________________________
 Time       :  65.000
**********************************************

>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : System_Monitor
 rule   : Sonar_Critical
 level  : low
 action : Pass_info_to_Equip_Assessor
 time   :  65.265


>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : System_Monitor
 rule   : Equipment_Status_Assessment
 level  : Assessment
 action : Assessing_Status
```

time : 65.426

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Overall_Mission
rule : Overall_Mission_Assessor
level : High
action : ContinueMission_with_restrictions
time : 65.596

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Overall_Mission
rule : Continue_mission_restricted_initial
level : Assessment
action : Note-time-of-status-change
time : 65.754

****
Event Number : 7

Description : rudder_has_failure_reading

Time : 68.000
****

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Maneuvering
rule : Maneuvering_Status_Assessment
level : maneuvering-assessment
action : change-overall-maneuvering-status
time : 68.303

>>>>>>>>>>>> Decision <<<<<<<<<<<<
type : Overall_Mission
rule : Overall_Mission_Assessor
level : High
action : Abort_mission
time : 68.470

```
>>>>>>>>>>>> Decision <<<<<<<<<<<<
 type   : Overall_Mission
 rule   : Abort_Mission
 level  : Low
 action : lock_status_and_replan_route_to_abort_rendezvous
 time   :   68.633


>>>>> Shutting Down for Dynamic Recovery <<<
>>>>> Transponder will function for 2 hours <<<
5761 rules fired        Run time is 72.20900000000074 seconds
79.78229860543618 rules per second
18 mean number of facts (29 maximum)
2 mean number of activations (12 maximum)
CLIPS> (dribble-off)
```

INITIAL DISTRIBUTION LIST

	No of Copies
1. Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2. Dudley Knox Library Code 052 Naval Postgraduate School Monterey, California 93943-5002	2
3. Chairman, Code CS Department of Computer Science Naval Postgraduate School Monterey, California 93943-5002	2
4. Dr. Y. Lee, Code CS/Le Department of Computer Science Naval Postgraduate School Monterey, California 93943-5002	7
5. Chairman, Code 69 Hy Department of Mechanical Engineering Naval Postgraduate School Monterey, California 93943-5002	1
6. LT W. P. Wilkinson, USN Department Head Class 121 Surface Warfare Officer School Command NETC Newport, Rhode Island 02841-5012	2

Thesis
W58833 Wilkinson
c.1 Mission executor for
an autonomous underwater
vehicle.

Thesis
W58833 Wilkinson
c.1 Mission executor for
an autonomous underwater
vehicle.

AMERICAN BINDERY
10
9-93

DUDLEY KNOX LIBRARY
3 2768 00018426 1